NEW MILFORD PUBLIC SCHOOLS
New Milford, Connecticut

AP Computer Science

June 2016

Approved by BOE December 2016

New Milford Board of Education

David Lawson, Chairperson
Bill Dahl, Vice Chairperson
Wendy Faulenbach, Secretary
Tammy McInerney, Assistant Secretary
Angela Chastain
Robert Coppola
David Littlefield
Brian McCauley
J.T. Schemm

Superintendent of Schools
Mr. Joshua Smith

Acting Assistant Superintendent
Ms. Alisha DiCorpo

Authors of Course Guide
Shana Bergonzelli-Graham

New Milford’s Mission Statement

The mission of the New Milford Public Schools, a collaborative partnership of students, educators, family and community, is to prepare each and every student to compete and excel in an ever-changing world, embrace challenges with vigor, respect and appreciate the worth of every human being, and contribute to society by providing effective instruction and dynamic curriculum, offering a wide range of valuable experiences, and inspiring students to pursue their dreams and aspirations.

Advanced Placement Computer Science

The focus of this course will be on problem-solving and developing skills in logic and reasoning through writing and analyzing programs based on computer theory and real-life scenarios. The goal is to be able to break down problems analytically and identify what is needed to develop an efficient and correct solution. One important skill is to be able to readily identify common algorithms and data structures and easily apply them to new problems or situations. We will explore how an object-oriented language lends itself to code reuse, readability and flexibility.

Java will be the vehicle for learning these skills, but the goal is to teach them so that students will be able to interpret and understand computer code in any object-oriented language. Code will be written and analyzed as small stand-alone programs as well as in the context of a larger program with many interacting objects and classes. Emphasis will also be placed on the interplay of hardware and software within a computer, important technical terminology and the proper and ethical use of computers.

Lesson Plan for Classroom Teachers

Teacher Shana Bergonzelli-Graham _____Grade Level_____10-12

Lesson Title: What is a Java Object?

Content Standards: Identify one or two **primary** content standards, including CCSS that this lesson is designed to help students attain.

CCSS.Math.Practice.MP1 Make sense of problems and persevere in solving them.

CCSS.Math.Practice.MP2 Reason abstractly and quantitatively.

AP Computer Science Topic Outline

IIB1:
Programming constructs: primitive types vs. objects

IA2:
Apply data abstraction and encapsulation.

IIA1a:
Methodology:Object Oriented development

IIB2c:
Class declarations

Literacy through the Content Area: If you will be using any strategies for teaching literacy in the content area, describe your plan.

Identification and definition of new vocabulary used within the reading. Use of target vocabulary in context.

Placement of Lesson within Broader Curriculum/Context: Where does this lesson fall within the sequence of the larger content standards or curriculum? Is it at the beginning, middle or end of a sequence of lessons/or a unit leading to attainment of the content standards? How will the outcomes of this lesson and student learning affect subsequent instruction?

This lesson is the first out of five in a unit on the features of Java classes and objects. It immediately follows a test on loops and conditionals. This lesson is one of the most important lessons taught in this class because it introduces the main benefit of the Java programming language: the ability for information to be stored in the form of objects that are derived from a class template. It is often difficult for students to make the leap from the object (tangible) to the abstract (class). Objects are very important in ensuring code reusability and organization.

Learner Background: Describe the students' prior knowledge or skill, and/or their present level related to the learning objective(s) and the content of this lesson (using data from pre-assessment as appropriate).

Students have homework that was assigned the night prior to this lesson where they read a chapter in the book that outlines the content in this lesson. After completing the reading, they are given a self-evaluative checklist that lets them self-assess their strengths and weaknesses with the material (attached). It is collected and the responses are quickly skimmed while the students are working in groups during the lesson. This helps better target those students that have deficiencies with the material.

Students have also gone on a scavenger hunt to identify objects within the school and try to determine properties/attributes and behaviors that these objects have in common so the students can eventually define a Java class as a group that emulates the objects they found.

Objective(s) for Lesson: Identify specific and measurable learning objectives/purpose for this lesson.

Students will :

- Understand how an object's properties or attributes are stored within instance variables in a class.
- Evaluate how methods define what objects of class can do and how methods manipulate data within an object of a class.
- Understand the purpose of a constructor.
- Create a class from scratch given the properties of the objects of the class (instance variables) and what objects of the class should do(methods).
- Instantiate an object of a class within another object—explain connection to constructor.
- Understand the importance of data encapsulation and how accessor/mutator methods facilitate this.

Assessment: How will you ask students to demonstrate mastery of the learning objective(s)? *Attach a copy of any assessment materials you will use, along with assessment criteria.* What data or evidence of student learning will be collected through the assessment?

Formative Assessments:

- What is an object? Discussion of object scavenger hunt results and set up project and file to transfer scavenger hunt classes/objects to code. How is an object different from a primitive data type?
- Guided Practice: Counter and TestCounter Classes—identify constructor, instance variables and accessor/mutator methods.
- Inquiry-based learning coding assignment (individual or pairs): Shapes project
- Coding: Implementation of Scavenger hunt class
- Lab Exercises: Student class, Coins class, Clock Class, Cash Register class.

Summative Assessments:

- Project: Magpie project (cumulative project: due before Christmas break)
- Quiz on material in lesson(early December)
- Third unit test (Early January)

Materials/Resources: List the materials you will use in each learning activity including any technological resources.

Activity 1(initiation: scavenger hunt results):

- SmartBoard software for visual display of activity.
- Internet-enabled computers.
- BlueJ Development Environment software for class setup

Activity 2(Guided practice with Counter and TestCounter code examples):

- SmartBoard software.
- Internet-enabled computers.
- BlueJ Development Environment software
- Counter.java and TestCounter.java class printouts and soft copies (attached).
- Tally Counter manipulative

Activity 3: Inquiry-Based Learning: Shapes program

- Internet-enabled computers.
- BlueJ Development Environment software
- Shapes project handout and organizer

Activity 4: Coding: Creation of Scavenger Hunt object and test class:

- Internet-enabled computers.
- BlueJ Development Environment
- Scavenger Hunt handout/organizer and project setup from guided practice.

Activity 5: Exit Ticket : Lesson and Self-Evaluation

- Exit Ticket Handout

Lesson Development/Instructional Strategies

- Identify the instructional grouping/s (whole class, small groups, pairs, individuals) you will use in each lesson segment and approximate time frames for each.

  Lesson Segment 1(initiation: object scavenger hunt results): whole class (5-10 minutes)
  Lesson Segment 2(Counter class guided practice and discussion): whole class (10-15 min)
  Lesson Segment 3(Inquiry-based learning: Shapes project): pairs (30-40 min)
  Lesson Segment 4(Scavenger hunt class/object Coding Assignment): individual (30 min)—will overlap into next class.
  Lesson Segment 5(Exit Ticket: Lesson and Self-Evaluation): individual (5 min)

- Describe what instructional strategies you will use and the learning activities in which students will be engaged in order to gain the key knowledge and skills identified in the learning objective(s). This may also include a description of how you will *initiate* (set expectations for learning and purpose) and *close* (understanding the purpose) the lesson.

This lesson plan was structured using the constructivist 5 Es model developed by Biological Science Curriculum Study (Engage, Explore, Explain, Elaborate, and Evaluate).

Before beginning a lesson, the class website is updated with the objectives for the day, the agenda, and list of assignments that the students need to turn in. This sets the tone for the lesson, assists students who need help with organization, and gives students clear expectations of what will happen in the lesson, as well as the items for which they are responsible.

Lesson Segment 1: Engage (initiation)

- Teacher initiates lesson by discussing how classes and objects are the topic for the day and why classes and objects are important in Java and programming overall.
- Teacher then defines what "object oriented" programming is.
- Teacher then instructs students to take out their scavenger hunt organizers and mentions that we are going to take these tangible objects and create classes for them in code.
- Teacher displays an organizer on the Smart board that designates a spot for the class name, specific instances of that class that were observed (objects), attributes or properties of that class (instance variables) and actions that the objects of the class can perform (methods). Teacher will model how to do this with a Locker class and specific locker objects.
- Students are encouraged to share the objects that they found on their scavenger hunt and the attributes and actions of the objects. Using this information, students will fill in the organizer.
- Teacher will also use this as an opportunity to point out the difference between variables that are primitive data types and variables that are objects.

Lesson Segment 2: Explore

- Teacher will demonstrate the tally counter tangible and invite students to pass it around and describe what information it needs to store and what it does. Teacher will write these descriptions on the board.
- Teacher will begin a project in BlueJ and create the TallyCounter class. These actions will be displayed on the Smart board.
- As a group, students will to define the TallyCounter class, and the teacher will pause to identify the constructor, default constructor, instance variables and methods. Students will answer questions to create the code for the class incrementally.
- Once the methods and instance variables have been defined, teacher will ask the class to examine each method to ensure that it does what it needs to do and add code if it is determined that it is necessary. One example is that the add method needs to check for whether the count is at 999, display a message and reset the counter if it is. What coding structure can we use to determine this?
- Teacher will explain what accessor and mutator methods are and invite the students to identify what methods are accessors and mutators in the TallyCounter class.
- Once the class definition is complete, the whole class will create a test class as a group and instantiate two TallyCounter objects. Teacher will be careful to explain the distinction between object instantiation and variable declaration and initialization. Teacher will explain that a test class allows us to understand and debug the behavior of classes. Teacher will invite students to make the distinction between the class that was defined earlier and the objects that are now

being created from the class. We will use the test class to demonstrate the functionality for the TallyCounter class and determine if all aspects of the class can be tested satisfactorily. What about when the TallyCounter needs to be reset? What programming structure can we use to add one to the counter 999 times?

- Teacher will also demonstrate that BlueJ has a built-in feature that emulates a test class and demonstrate this feature to the students.

Lesson Segment 3: Explain

- Teacher instructs the students that they will be working in pairs for the next assignment and assigns pairs. Teacher then passes out a handout for the Shapes Coding Assignment.
- Teacher then guides students to download the project indicated on their Shapes handout from the class website. This project will need to be unzipped. Once the project is unzipped, it can be opened in BlueJ.
- This assignment guides students to create objects as shapes on a canvas and allows them to manipulate the objects by calling methods on them to change their properties. It also allows them to make the connection between the class, which is a template for an object and the individual objects themselves.
- Students will notice that each object that they create is separate and has its own properties that define it and make it different from other objects of the same class. However, what the objects have in common defines them as members of the same class. They will need to state this relationship clearly, and the teacher will use their responses to check for understanding. This assignment also asks the students leading questions so they can form their own perceptions regarding the material and allows them to be creative.

Lesson Segment 4: Elaborate

- Once the student has completed the Shapes assignment they will be independently taking their Scavenger Hunt organizer and creating a Java class that emulates the behavior of the class they found during their scavenger hunt.
- They will also be independently creating a test class and instantiating objects of the class they created in their test class.

Lesson Segment 5: Evaluate(closure)

- Teacher facilitates a quick question/answer and discussion session to check for understanding on key concepts.
- Teacher passes out the exit tickets and informs students that they will be filling this out to evaluate the clarity of the lesson in addition to self-evaluating their comprehension of the material.
- Teacher reminds students to check the website for the list of homework assignment that needs to be completed for next class.

*This quick exit ticket allows students to self-assess their understanding of the material and helps inform any additional practice or remediation that can be planned into future lessons. It also allows for reflection on presentation of the lesson and identification of any strengths and weaknesses. This

allows the opportunity to make any necessary modifications for the next time this lesson is taught and assists in improving practice.

Pacing Guide

Unit #	Title	Weeks	Pages.
1	Java Programming Basics	6 Weeks	12-16
	Ethical Computer Use and Handware/Software	1	
	Problem Solving Process and Algorithms	½	
	Program Structure and Syntax	1	
	Commenting, Formatting and Javadoc program Documentation	½	
	Variables, Output and User Input	1	
	String variables, String Operations and methods of the String class	1	
	Mathematical Operations, Mixed Data Types and the Math Class Methods	1	
2	The Conditional and Repetitive Structures with Arrays	9 Weeks	17-22
	Logical and Relational Operators	1	
	Decision Making in Programs (if-else)	2	
	Repetition in Programs (for and while loops)	2	
	Arrays and Using Arrays with loops	2	
	2-Dimensional Arrays	1	
	Sequential and Binary Searching	1	
3	Class Structure, Object Arrays and the ArrayList Object	4 ½ Weeks	23-27
	Class Structure, Encapsulation and Accessor/Mutator methods	2	
	Static Methods and Variables	1	
	Arrays of objects and the ArrayList class	1	
	Wrapper Classes and the toString method.	½	
4	Inheritance, Abstract Classes, Interfaces and Polymorphism	4 ½ Weeks	28-32
	Inheritance and method overriding	1 ½	
	Abstract Methods and Classes	1 ½	
	Interfaces and Polymorphism	1	
	Arrays of Objects and the ArrayList Class Revisited	½	
5	Recursion, Sorting and Algorithm Analysis	4 Weeks	33-36
	Common recursive algorithms and identification of base cases and recursive calls	1	

New Milford Public Schools

Committee Member(s): Shana Bergonzelli Unit Title: Java Programming Basics	Course/Subject: AP Computer Science Grade Level: 11-12 # of Weeks: 6

Identify Desired Results

Common Core Standards and College Board Topic Outline Standards

- RST4: Determine the meaning of symbols, key terms and other domain-specific words and phrases as they are used in a specific scientific or technical context.
- RI7: Integrate and evaluate multiple sources of information presented in different media or formats as well as in words in order to address a question or solve a problem.
- **CCSS.Math.Practice.MP1** Make sense of problems and persevere in solving them.
- **CCSS.Math.Practice.MP2** Reason abstractly and quantitatively.
- **CCSS.ELA-LITERACY.RST.11-12.3**
  Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.

- **I Object-Oriented Program Design**

The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process needs to be based on a thorough understanding of the problem to be solved.

- A. Program design
  - 1. Read and understand a problem description, purpose, and goals.

- **II. Program Implementation**

The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

- B. Programming constructs
  - 1: Primitive Types
  - 2: Declarations
    - a. Constant declarations
    - b. Variable declarations
    - c. Class declarations
    - e. Method declarations
    - f. Parameter declarations
  - 3: Console Output (System.out.print/println)
  - 4. Control
    - a. Methods
    - b. Sequential

- **III. Program Analysis**

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

- B. Debugging

        - 1. Categorize errors: compile-time, run-time, logic.
        - 2. Identify and correct errors.
    - H. Numerical representations and limits
        - 2. Limitations of finite representations (e.g., integer bounds, imprecision of floating-point representations, and round-off error)

- **IV. Standard Data Structures**

Data structures are used to represent information within a program. Abstraction is an important theme in the development and application of data structures.

- A. Simple data types (int, boolean, double)

- **VI. Computing in Context**

An awareness of the ethical and social implications of computing systems is necessary for the study of computer science. These topics need not be addressed in detail but should be considered throughout the course.

- A..System reliability
- B..Privacy
- C..Legal issues and intellectual property
- D. Social and ethical ramifications of computer use

Enduring Understandings Generalizations of desired understanding via essential questions (Students will understand that …)	**Essential Questions** Inquiry used to explore generalizations
• Computer Programming is about solving problems. • It is important to thoroughly understand the problem one is trying to solve and clarify assumptions before going about solving it. • Java programs are structured a certain way so the compiler can understand the code. • Programs should be written in such a way that they can be understood by programmers other than the author. • Planning and designing code is an important step of the problem solving process. • Computer programs are sequential (one statement follows another). • Information is represented in programs via data structures. • Data structures can represent different kinds of data (numerical, text, etc). • Abstraction of the data structures in a computer programs means that	• What is the problem solving process? • What is an algorithm? • How is the creation of an algorithm related to the problem solving process? • How can we analyze a problem statement to synthesize important information and clarify assumptions? • How can algorithms lend themselves to reusability? • Why are coding conventions important to follow? • What is the purpose of comments in a computer program? • What are some strategies I can use to plan code (flowcharts, pseudocode, etc)? • How are Java programs structured? • In what order are statements in a Java program executed? • How do I display information to the screen in a Java program? • How do I retrieve information from the user in a Java program?

the structures themselves can stand for any relevant value. • An important part of programming is identifying and fixing errors in code (debugging). • Input and output are important concepts to consider when determining the flow of information. • Computer programmers should always consider the ethical and social implications related to technology use.	• What are variables and what kind of information can be stored in them? • What does syntax mean? • What are some common syntax errors in Java code and how do I fix them? • How are text, numbers and other kinds of information represented in Java? • How is data manipulated via various operations in Java (e.g. concatenation for Strings, mathematical operations for numbers, logical operations for booleans, etc)?

Expected Performances
What students should know and be able to do

Students will know the following:

- Solving problems is a process.
- Statements in programs are executed sequentially.
- Just like in languages we use for verbal communication, Java has rules for how a program is written called syntax.
- Syntax errors occur due to improper use of the Java language.
- Variables are created in such a way as to only hold a specific kind of data.
- Variables can stand for any value that conforms to their data type.
- An important part of being an effective programmer is to only use technology in a positive way.
- Mathematical expressions in Java follow the PEMDAS order of operations rule.
- Integer division will truncate the decimal component of the result.
- Use a decimal in mathematical expressions to promote the result to a decimal.

Students will be able to do the following:

- Create a Java program with a main method that compiles
- Document and format their code so it is easily understandable by other programmers.
- Understand a problem statement and synthesize the necessary information needed to code a solution.
- Create and use String, double and int variables in programs meaningfully.
- Output information to the screen.
- Input information from the user in various formats (Strings, ints, doubles).
- Use mathematical expressions meaningfully in programs to solve problems.
- Use casts on data types correctly.
- Identify and fix common syntax errors.

Character Attributes

- Perseverance
- Integrity

- Responsibility
- Honesty

Technology Competencies

- Students collaborate with peers or others to solve problems and to develop solutions using technology tools and resources.
- Students identify and define object-oriented programming terminology.
- Students identify and explain programming structures.

Develop Teaching and Learning Plan

Teaching Strategies:

- Clarification of problem solving process and its importance in computer science.
- Demonstration of coding concepts through analysis of worked examples.
- Guided Practice and Demonstration
- Vocabulary in context.
- Using visual aids like flowcharts to demonstrate the flow of control in programs.
- Flipped Learning (Students read material and complete an inductive assignment on the reading in order to come into class with questions, misconceptions, etc.)
- Discussion of questions /misconceptions re: content read.
- Use various sized boxes to demonstrate how data is stored in variables of different data types.
- Demonstrate to students the expected behavior of the program so they can self-correct their programs in progress.
- Leveled assignments are offered for students who need remediation or more challenge.
- Circulate and assist while students are completing assignments.
- Assign certain advanced students to be “mentors” to struggling learners.

Learning Activities:

- Students complete a set of inductive learning Lab Exercises on the 7 lessons taught in the unit.
- Students create flowcharts to plan out the logic of their programs before coding them.
- Students pair and use the problem solving process to work on a program that outputs information in a Graphical User Interface format (HelloGui Program).
- Students complete vocabulary worksheets for the 7 lessons taught in the unit.
- Students answer AP style multiple choice questions.
- Students apply their knowledge of the formula for generating random numbers to complete the RandomDots coding assignment.
- Students pair/group to discuss and correct completed work.
- Students use various debugging strategies such as the Jeliot program, output statements and breakpoints to determine errors in code.
- For extra help and remediation, students watch videos based on content taught in class
- Students complete exit tickets.

Assessments

Performance Task(s) Authentic application to evaluate student achievement of desired results designed according to GRASPS (one per marking period)	**Other Evidence** Application that is functional in a classroom context to evaluate student achievement of desired results
Goal: Completion of a vote tallying program (Poll Project) Role: Programmer and Tester Audience: Users of the Software Situation: Students are given an incomplete program that is intended to record and tally votes for a student election. There are three candidates for whom votes are supposed to be displayed as a pie chart. Students work in pairs to apply mathematical programming concepts to compute the arc size that represents the votes for each candidate and display the arc with the color that corresponds to the correct candidate. Problem solving, random number generation, mathematical operations, data type casting and error-checking are skills that are assessed Product or Performance: A completed, functional Poll program with no errors Standards for Success: According to Rubric.	• Quizzes (1 every two lessons) • Unit Test • Various coding-based Performance Assessments (detailed above) • Homework completion • AP-style Multiple Choice questions • Poll Project

Suggested Resources

- Blue J IDE: www.bluej.org
- Bradley Kjell's Online Introduction to Java Course—Central Connecticut State University: http://chortle.ccsu.edu/CS151/cs151java.html
- Cook, Charles E. Blue Pelican Java. College Station: Virtualbookworm.com, 2005. Print.
- Litvin, Maria, and Gary Litvin. Java Methods: Object-oriented Programming and Data Structures. Andover, MA: Skylight Pub., 2011. Print.
- College Board Computer Science A AP Central Website: http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html
- Lew Tube Programming Tutorial Videos: http://www.thecubscientist.com/APCS/indexAPCS_HW.html
- Various Teacher-Created assignments
- College Board AP® Computer Science A Quick Reference Sheet

New Milford Public Schools

Committee Member(s): Shana Bergonzelli Unit Title: The Conditional and Repetitive Structures and Arrays	Course/Subject: AP Computer Science Grade Level: 11-12 # of Weeks: 8-9

Identify Desired Results

Common Core Standards and College Board Topic Outline Standards

- RST4: Determine the meaning of symbols, key terms and other domain-specific words and phrases as they are used in a specific scientific or technical context.
- RI7: Integrate and evaluate multiple sources of information presented in different media or formats as well as in words in order to address a question or solve a problem.
- **CCSS.Math.Practice.MP1** Make sense of problems and persevere in solving them.
- **CCSS.Math.Practice.MP2** Reason abstractly and quantitatively.

- **II. Program Implementation**

The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

  - B. Programming constructs
    - 4. Control
      - c. Conditional
      - d. Iteration

- **III. Program Analysis**

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

- A. Testing
  - 2. Identify boundary cases and generate appropriate test data.
- B. Debugging
  - 3. Employ techniques such as using a debugger, adding extra output statements, or hand-tracing code.
- G. Analysis of algorithms
  - 2. Exact calculation of statement execution counts

- **IV. Standard Data Structures**

Data structures are used to represent information within a program. Abstraction is an important theme in the development and application of data structures.

  - C. Lists
  - D. Arrays

- **V. Standard Algorithms**

Standard algorithms serve as examples of good solutions to standard problems. Many are intertwined with standard data structures. These algorithms provide examples for analysis of program efficiency.

  - A. Operations on data structures previously listed
    - 1. Traversals
    - 2. Insertions
    - 3. Deletions

- B. Searching
  - 1. Sequential
  - 2. Binary

Enduring Understandings Generalizations of desired understanding via essential questions (Students will understand that …)	**Essential Questions** Inquiry used to explore generalizations
• It is essential that programs make decisions in order to implement necessary functionality. • Testing programs is a necessary part of ensuring that they work correctly in all cases. • It is important to identify and use various strategies such as breakpoints and output statements for debugging and testing code. • Breaking code down in to smaller parts (procedural decomposition) allows for code reusability and lends itself to algorithms that are easier to understand and use. • Repetition is a concept that allows programs to perform tasks quickly in a small amount of code. • Arrays allow large amounts of same-type data to be stored under one variable name for easier access, traversal and item retrieval.	• How do I use the conditional structure to make decisions in programs? • When it is appropriate to use the conditional structure to solve problems? • Why are indentation rules especially important when coding conditionals and loops? • How can I use logic to code my conditional structure so the outcome is correct? • When it is appropriate to use the repetitive structure to solve problems? • How do I identify appropriate circumstances for different kinds of loops? • What are some pitfalls that occur when coding loops? • How do I identify scenarios where it is appropriate to use procedural decomposition to break code down into methods? • How do I analyze what a method needs to do in order to determine parameters and return type? • How do I call a method from another method? • What are some best practices for manipulating items in an array?

Expected Performances
What students should know and be able to do

Students will know the following:

- Decisions made in programs may have one or more branches.
- Decisions made in programs can also be nested inside each other.
- Boolean expressions resolve to ONLY true or false.
- The result of a Boolean expression can be stored in a boolean variable.
- The relational operators for Boolean expressions (<, >, <=, >=, ==, !=) and their meanings.
- The logical operators for Boolean expressions (&&, || and !) and their meanings.

- The condition is written as a Boolean expression and it is what controls the outcome of the decision.
- A loop is how programs perform actions repeatedly to solve problems.
- Loops need to be constructed so that they eventually terminate.
- An endless loop is a loop that dies not terminate
- Three common kids of loops are counting loops, sentinel controlled loops and result controlled loops.
- An array is a data structure that implements a list of differing items that share the same data type.
- Arrays are indexed starting at zero, not one.
- The size of an array is stored in the array's .length variable.
- Loops are useful for traversing arrays to solve problems.
- Two dimensional arrays are declared in a similar manner to arrays and can be described as a matrix.

Students will be able to do the following:

- Simplify Boolean expressions using DeMorgan's Law and other strategies.
- Evaluate Boolean expressions.
- Identify equivalent Boolean expressions.
- Code decisions in programs using the if-else structure.
- Code a condition using a syntactically correct if-else structure that is appropriate in the context of the problem given.
- Read a problem statement that includes a decision and code a solution using the if-else structure and relational/logical operators as necessary.
- Code a program that uses repetition to solve a problem.
- Code syntactically correct for and while loops.
- Use a formula to determine the number of iterations that are performed by a counting loop.
- Analyze a problem to determine whether it is more appropriate to use a counting loop, a sentinel controlled loop or a result controlled loop.
- Declare and use arrays correctly within a program.
- Code programs that traverse an array to search for an item (using both sequential and binary search algorithms).
- Code programs that manipulate items within an array (addition, insertion and removal of items).
- Identify OBOBs in loops (off by one bugs).
- Analyze programs that use loops or conditionals to determine their behavior.
- Code programs that correctly declare two dimensional arrays of various sizes.
- Code programs that use nested loops to traverse two-dimensional arrays.
- Hand trace programs using a chart to determine a program's behavior.
- Identify the different parts of a method signature. (access level, return type, name and parameter list)
- Create methods that have appropriate names, return types and parameter lists and call them correctly from other methods.
- Identify common or repeated code and consolidate it into methods that can be easily called from other parts of the program.

Character Attributes

- Perseverance
- Cooperation

Technology Competencies

- Students collaborate with peers or others to solve problems and to develop solutions using technology tools and resources.
- Students identify and define object-oriented programming terminology.
- Students identify and explain programming structures.

Develop Teaching and Learning Plan

Teaching Strategies:

- Demonstration of coding concepts through analysis of worked examples.
- Guided Practice and Demonstration
- Vocabulary in context.
- Using visual aids like flowcharts to demonstrate the flow of control in programs.
- Use manipulatives and visual aids such as a children's linked train car toy or a linear chart with slots and movable entries to demonstrate how items are stored and manipulated within an array.
- Flipped Learning (Students read material and complete an inductive assignment on the reading in order to come into class with questions, misconceptions, etc)
- Demonstrate best practice for hand tracing code (identification of variables, use of a chart, crossing out old entries).
- Demonstrate to students the expected behavior of the program so they can self-correct their programs in progress.
- Discussion of questions /misconceptions re: content read.
- Leveled assignments are offered for students who need remediation or more challenge.
- Circulate and assist while students are completing assignments.
- Assign certain advanced students to be "mentors" to struggling

Learning Activities:

- Students complete a set of inductive learning Lab Exercises on the 7 lessons taught in the unit.
- Students create flowcharts to plan out the logic of their programs before coding them.
- Students use the problem solving process to work on a program that uses decision making to simulate an elevator (Elevator Program).
- Students complete vocabulary worksheets for the 7 lessons taught in the unit.
- Students answer AP style multiple choice questions.
- Students complete code analysis. worksheets
- Students complete hand tracing problem set worksheets. Students complete bug identification worksheets
- Students apply their knowledge of the repetitive structure to create program that uses a counting loop to simulate printing out a deck of cards. (HouseOfCards program)
- Students complete a program that includes various methods that manipulate array items. Students demonstrate that the code works by calling the methods within a separate test class (ArrayManip Program).
- In pairs, students complete a graphical applet that constructs a String array of various fortunes and uses the Math.random function to randomly pick

learners.	a fortune (Fortune Teller Applet) • Students pair/group to discuss and correct completed work. • For extra help and remediation, students watch videos based on content taught in class. • Students complete exit tickets.

Assessments

Performance Task(s) Authentic application to evaluate student achievement of desired results designed according to GRASPS (one per marking period)	**Other Evidence** Application that is functional in a classroom context to evaluate student achievement of desired results
Goal: In groups, students will modify a program that simulates a working chatbot that will ask the user questions and respond appropriately. Role: Programmer/Tester. Audience: Users of the program. Situation: This lab will take strings entered by the user, parse them for recognizable information, and give an appropriate response according to the information found. Students will be exposed to the Java API for Strings and learn how to usefully incorporate the conditional and repetitive structures in programs. Product or Performance: A functional Magpie program that interacts with the user and responds to information given. Standards for Success: Completed, working program graded via rubric. Associated Multiple Choice and Free Response questions.	• Quizzes (1 every two lessons) • Unit Test • Various coding-based Performance Assessments (detailed above) • Homework completion • AP-style Multiple Choice questions • AP-style Free-Response Questions • Magpie Project

Suggested Resources

- Blue J IDE: www.bluej.org
- Bradley Kjell's Online Introduction to Java Course—Central Connecticut State University: http://chortle.ccsu.edu/CS151/cs151java.html
- Cook, Charles E. Blue Pelican Java. College Station: Virtualbookworm.com, 2005. Print.
- Litvin, Maria, and Gary Litvin. Java Methods: Object-oriented Programming and Data

Structures. Andover, MA: Skylight Pub., 2011. Print.

- College Board Computer Science A AP Central Website: http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html
- Lew Tube Programming Tutorial Videos: http://www.thecubscientist.com/APCS/indexAPCS_HW.html
- Various Teacher-Created assignments
- College Board AP® Computer Science A Quick Reference Sheet
- College Board AP® Computer Science A Magpie Chatbot Lab Teacher's Guide
- College Board AP® Computer Science A Magpie Chatbot Lab Student Guide

New Milford Public Schools

Committee Member(s): Shana Bergonzelli Unit Title: Class Structure, Object Arrays and the ArrayList object	Course/Subject: AP Computer Science Grade Level: 11-12 # of Weeks: 4 ½ to 5

Identify Desired Results

Common Core Standards and College Board Topic Outline Standards

- **I. Object-Oriented Program Design**
  The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process needs to be based on a thorough understanding of the problem to be solved.
    - A. Program design
      - 1. Read and understand a problem description, purpose, and goals.
      - 2. Apply data abstraction and encapsulation.
      - 3. Read and understand class specifications and relationships among the classes ("is-a," "has-a" relationships).
    - B. Class design
      - 1. Design and implement a class.
      - 2. Choose appropriate data representation and algorithms.
      - 3. Apply functional decomposition.

- **II. Program Implementation**

The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

- A. Implementation techniques
  - 1. Methodology
    - a. Object-oriented development
    - b. Top-down development
    - c. Encapsulation and information hiding
    - d. Procedural abstraction
- B. Programming constructs
  - 1. Primitive types vs. objects
  - 2. Declaration
    - c. Class declarations
    - e. Method declarations
    - f. Parameter declarations
- C. Java library classes (included in the AP Java subset)

- **III. Program Analysis**

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

- A. Testing
  - 1. Test classes and libraries in isolation.
  - 2. Identify boundary cases and generate appropriate test data.
  - 3. Perform integration testing.
- B. Debugging
  - 2. Identify and correct errors.

        - 3. Employ techniques such as using a debugger, adding extra output statements, or hand-tracing code.
    - C. Understand and modify existing code
    - D. Extend existing code using inheritance
    - E. Understand error handling
        - 1. Understand runtime exceptions.

- **IV. Standard Data Structures**

Data structures are used to represent information within a program. Abstraction is an important theme in the development and application of data structures.

- B. Classes
- C. Lists
- D. Arrays

Enduring Understandings Generalizations of desired understanding via essential questions (Students will understand that …)	**Essential Questions** Inquiry used to explore generalizations
• Code should be designed in such a way as to promote reusability. • Object oriented programming lends itself to code reusability. • A class is a template that defines the attributes and behaviors of objects created from it. • It is important for objects to maintain private data in order to minimize data breaches and incorrect information. • Static methods and variables allow for maintenance of information and functionality common to all objects of a class. • Objects can be stored in arrays and accessed the same way as primitive data types.	• What is the difference between a class and an object? • How are classes and objects related? • Why is encapsulation (information hiding) important to data integrity? • How do accessors and mutators allow class data to be used and modified robustly? • How are objects created from a class template? • How are methods called on objects of the class? • How do methods access object data and perform actions? • How can I solve programming problems using object oriented concepts?

Expected Performances
What students should know and be able to do

Students will know the following:

- A class summarizes everything that describes an object (attributes) and what an object can do (behaviors).
- Attributes are expressed as instance variables.
- Behaviors are expressed as methods.
- The difference between public and private methods and variables.
- Encapsulation refers to the practice of keeping object data private.
- Accessor methods (getters) allow client classes to access object data. Mutator methods (setters) allow client classes to change object data.
- Instantiation refers to the process of creating a variable of an object type. Instantiation involves setting aside computer memory where the object's data is

stored.

- A constructor is a special method that is invoked when an object of a class is instantiated. The constructor initializes the object's data.
- Static methods allow functionality that does not access object data but is common to all objects of the class to be implemented.
- Static variables are common to all objects of the class to be implemented.
- Arrays of objects are created and used the exact same way as primitive data type arrays.
- Wrapper classes are classes that wrap int and double values. They are named Integer and Double. They define special methods for using and converting the values.
- The this keyword is used with dot notation within a class to explicitly refer to instance variables and methods belonging to the class.
- Object variables that have not been instantiated are referred to by the keyword null.
- ArrayLists have similar functionality to arrays but can only contain objects.
- If you want an ArrayList of numbers, you need to convert the numbers to objects of the Integer and/or Double classes.
- ArrayLists are objects themselves and have a special API for accessing, adding, inserting and removing items.

Students will be able to do the following:

- Identify the different parts of a class.
- Identify accessor and mutator methods.
- Use the this keyword within a class.
- Differentiate between default and multi parameter constructors.
- Create default and multi parameter constructors that initialize all instance variables appropriately.
- Create a class that incorporates best practices for encapsulation by making all instance variables private and declaring public accessor and mutator methods.
- Declare, instantiate and use objects of a class within a client class appropriately.
- Use object oriented principles to solve problems.
- Create and manipulate arrays of objects.
- Call methods on objects in an array using dot notation.
- Code and use static methods and variables appropriately.
- Override the Java Object toString method in order to print out object data specific to each object.
- Invoke the toString method explicitly and implicitly using System.out.println.
- Instantiate ArrayLists and manipulate items within them using the given API.
- Use the Integer and Double wrapper classes with ArrayLists.

Character Attributes

- Perseverance
- Cooperation

Technology Competencies

- Students collaborate with peers or others to solve problems and to develop solutions using technology tools and resources.

- Students identify and define object-oriented programming terminology.
- Students identify and explain programming structures.

Develop Teaching and Learning Plan

Teaching Strategies:

- Demonstration of coding concepts through analysis of worked examples.
- Guided Practice and Demonstration
- Vocabulary in context.
- Review and discuss Scavenger Hunt findings as a class to correctly identify classes and objects found.
- Use Scavenger hunt as springboard to discuss attributes (items that can describe the class) of a class and behaviors (things the class can do).
- Use of tally counter manipulative to demonstrate a class that stores data (the number on the counter) and has behaviors (adding one to the counter, resetting the counter, turning over to zero when the counter reaches 999, etc) and various objects of that class that have different states.
- Use tally counter to demonstrate why encapsulation is important (other programs shouldn't be able to set counter value arbitrarily).
- Flipped Learning (Students read material and complete an inductive assignment on the reading in order to come into class with questions, misconceptions, etc)
- Discussion of questions /misconceptions re: content read.
- Leveled assignments are offered for students who need remediation or more challenge.
- Circulate and assist while students are completing assignments.
- Assign certain advanced students to be "mentors" to struggling learners.

Learning Activities:

- School-wide Scavenger hunt to identify classes and objects that can be classified.
- Students complete a set of inductive learning Lab Exercises on the 6 lessons taught in the unit.
- Students complete vocabulary worksheets for the 6 lessons taught in the unit.
- Students answer AP style multiple choice questions.
- Students answer AP style free response questions.
- Students complete a Shapes coding assignment that demonstrates object oriented properties.
- Students create a class and instantiate objects of that class in a test class. Accessor/Mutator methods will be discussed and implemented.(Student program).
- Students create an array of student objects in the test class and traverse the array searching for student objects with various attributes.
- Students complete a Coins class that simulates the most efficient way change can be divided up into coins.
- Students complete a classes that simulates bank accounts, clocks and cash registers.
- Students pair/group to discuss and correct completed work.
- For extra help and remediation, students watch videos based on content taught in class.
- Students use given rubrics to self-evaluate programs.
- Students complete exit tickets.

Assessments

Performance Task(s) Authentic application to evaluate student achievement of desired results designed according to GRASPS (one per marking period)	**Other Evidence** Application that is functional in a classroom context to evaluate student achievement of desired results
	• Quizzes (1 every two lessons) • Unit Test • Various coding-based Performance Assessments (detailed above) • Homework completion • AP-style Multiple Choice questions • AP-style Free-Response Questions • Java Garden Project

Suggested Resources

- Blue J IDE: www.bluej.org
- Bradley Kjell's Online Introduction to Java Course—Central Connecticut State University: http://chortle.ccsu.edu/CS151/cs151java.html
- Cook, Charles E. Blue Pelican Java. College Station: Virtualbookworm.com, 2005. Print.
- Litvin, Maria, and Gary Litvin. Java Methods: Object-oriented Programming and Data Structures. Andover, MA: Skylight Pub., 2011. Print.
- College Board Computer Science A AP Central Website: http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html
- Lew Tube Videos: http://www.thecubscientist.com/APCS/indexAPCS_HW.html
- Various Teacher-Created assignments

New Milford Public Schools

Committee Member(s): Shana Bergonzelli Unit Title: Inheritance, Abstract Classes, Interfaces and Polymorphism	Course/Subject: AP Computer Science Grade Level: 11-12 # of Weeks: 9-10

Identify Desired Results

Common Core Standards and College Board Topic Outline Standards

- **I. Object-Oriented Program Design**
  The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process needs to be based on a thorough understanding of the problem to be solved.
    - A. Program design
      - 1. Read and understand a problem description, purpose, and goals.
      - 2. Apply data abstraction and encapsulation.
      - 4. Understand and implement a given class hierarchy.
      - 5. Identify reusable components from existing code using classes and class libraries.
    - B. Class design
      - 1. Design and implement a class.
      - 2. Choose appropriate data representation and algorithms.
      - 3. Apply functional decomposition.
      - 4. Extend a given class using inheritance.
- **II. Program Implementation**

The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

- A. Implementation techniques
  - 1. Methodology
    - a. Object-oriented development
    - b. Top-down development
    - c. Encapsulation and information hiding
    - d. Procedural abstraction
- B. Programming constructs
  - 1. Primitive types vs. objects
  - 2. Declaration
    - c. Class declarations
    - d. Interface declarations
    - e. Method declarations
    - f. Parameter declarations
- C. Java library classes (included in the AP Java subset)

- **III. Program Analysis**

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

- A. Testing
  - 1. Test classes and libraries in isolation.
  - 2. Identify boundary cases and generate appropriate test data.
  - 3. Perform integration testing.

- B. Debugging
  - 2. Identify and correct errors.
  - 3. Employ techniques such as using a debugger, adding extra output statements, or hand-tracing code.
- C. Understand and modify existing code
- D. Extend existing code using inheritance
- E. Understand error handling
  - 1. Understand runtime exceptions.

- **IV. Standard Data Structures**

Data structures are used to represent information within a program. Abstraction is an important theme in the development and application of data structures.

- B. Classes
- C. Lists
- D. Arrays

Enduring Understandings Generalizations of desired understanding via essential questions (Students will understand that …)	**Essential Questions** Inquiry used to explore generalizations
• Inheritance allows for ease of code reuse and prevents code duplication and obscurity. • Abstract classes allow for compile-time safety by necessitating implementation of key functionality in derived classes. • Interfaces are important tools in assisting with standardization of code. • Polymorphism allows for a programmer to call methods on an object and be confident the methods perform correctly without knowing the underlying object type or understanding the method implementation. • The concept of polymorphism is useful when storing objects in arrays and ArrayLists.	• Why is inheritance useful and necessary? • How can we use the concept of inheritance to solve programming problems? • What is the benefit of declaring and using abstract classes? • What is the purpose of interfaces? • What are the benefits of polymorphism?

Expected Performances
What students should know and be able to do

Students will know the following:

- Inheritance allows for ease of code extension without duplication of common methods and instance variables.
- A class cannot inherit from more than one class (multiple inheritance dilemma). A class can implement many interfaces.
- The parent class is considered the superclass. All classes that extend the superclass are subclasses.

- The Java Object class is the implicit superclass of all Java classes.
- Subclasses do not inherit their superclass' constructors.
- Data and functionality that is common to all subclasses should be kept in the superclass and not reimplemented in the subclass.
- Method overriding refers to the practice of implementing a superclass' method of the same name and parameter list in the subclass. This is beneficial when the subclass has different or additional functionality in the method that does not exist in the superclass.
- The signature of a method depends only on the number, types, and order of its parameters but does not include the return type of the method. This is important for method overriding.
- Polymorphism allows for a superclass variable to hold a subclass object. The object "knows" it is of the subclass type, therefore when a method is invoked, it calls the subclass' version of the method.
- Subclass object variables cannot hold superclass objects.
- Abstract classes contain abstract methods. Concrete classes contain NO abstract methods.
- Abstract classes CANNOT be instantiated.
- Abstract methods are methods without a body that force implementation in concrete subclasses.
- Interfaces are classes that only contain abstract methods.

Students will be able to do the following:

- Create a subclass that appropriately extends the functionality of a superclass without code duplication.
- Create a subclass that overrides functionality in the superclass.
- Call methods and constructers of the superclass within a subclass appropriately by using the super keyword.
- Use correct syntax when creating classes that extend other classes.
- Declare, instantiate and use objects of superclasses and subclasses in a client class appropriately.
- Use the principle of polymorphism to declare object variables of the superclass type and instantiate them as the subclass.
- Analyze problems and solve them using the principles of inheritance and polymorphism.
- Create an abstract class that declares appropriate abstract methods.
- Extend an abstract class and appropriately implement its methods.
- Create an interface and use appropriate naming conventions and syntax.
- Implement the interface within another class.
- Use the Comparable Interface and implement the compareTo method appropriately.
- Use class casts appropriately.
- Use polymorphic principles to call methods on objects in an array and observe behavior.
- Create a large program utilizing arrays, ArrayLists, inheritance and polymorphism with many interconnected objects.

Character Attributes

- Perseverance
- Cooperation

Technology Competencies

- Students collaborate with peers or others to solve problems and to develop solutions using technology tools and resources.
- Students identify and define object-oriented programming terminology.
- Students identify and explain programming structures.

Develop Teaching and Learning Plan

Teaching Strategies:

- Demonstration of coding concepts through analysis of worked examples.
- Guided Practice and Demonstration
- Vocabulary in context.
- Smartboard is-a, has-a class/object classification activity to demonstrate composition and inheritance relationships.
- Teacher demonstrates the interplay between interfaces and class extension by using the example of a TV class that needs a cord connection interface.
- Flipped Learning (Students read material and complete an inductive assignment on the reading in order to come into class with questions, misconceptions, etc)
- Discussion of questions /misconceptions re: content read.
- Leveled assignments are offered for students who need remediation or more challenge.
- Circulate and assist while students are completing assignments.
- Assign certain advanced students to be “mentors” to struggling learners.

Learning Activities:

- Students complete a set of inductive learning Lab Exercises on the 4 lessons taught in the unit.
- Students complete vocabulary worksheets for the 4 lessons taught in the unit.
- Students answer AP style multiple choice questions.
- Students answer AP style free response questions.
- Students a SavingsAccount class that extends the previously completed BankAccount class to show best practice use of inheritance.
- Students create an array of objects in a test class that share a superclass. They then traverse the array calling the same method on each object to demonstrate the concept of polymorphism ((Monster Mash program).
- Students complete a coding exercise that implements the Comparable interface for Coins objects.
- Students pair/group to discuss and correct completed work.
- For extra help and remediation, students watch videos based on content taught in class.
- Students use given rubrics to self-evaluate programs.
- Students complete exit tickets.

Assessments

Performance Task(s) Authentic application to evaluate student achievement of desired results designed according to GRASPS (one per marking period)	**Other Evidence** Application that is functional in a classroom context to evaluate student achievement of desired results
Goal: Creation of a program that uses objects to simulate plants growing in a garden (Java Garden Project) Role: Programmer/Tester Audience: Users of the program Situation: In pairs, students create a program in sequential steps that simulates plants growing in a garden. Object oriented and inheritance concepts are stressed as well as the idea and usefulness of interfaces. Product or Performance: Working Java Garden project. Standards for Success: Graded via rubric.	• Quizzes (1 every two lessons) • Unit Test • Various coding-based Performance Assessments (detailed above) • Homework completion • AP-style Multiple Choice questions • AP-style Free-Response Questions • Java Garden Project

Suggested Resources

- Blue J IDE: www.bluej.org
- Bradley Kjell's Online Introduction to Java Course—Central Connecticut State University: http://chortle.ccsu.edu/CS151/cs151java.html
- Cook, Charles E. Blue Pelican Java. College Station: Virtualbookworm.com, 2005. Print.
- Litvin, Maria, and Gary Litvin. Java Methods: Object-oriented Programming and Data Structures. Andover, MA: Skylight Pub., 2011. Print.
- College Board Computer Science A AP Central Website: http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html
- Lew Tube Videos: http://www.thecubscientist.com/APCS/indexAPCS_HW.html
- Various Teacher-Created assignments

New Milford Public Schools

Committee Member(s): Shana Bergonzelli Unit Title: Recursion, Sorting and Algorithm Analysis	Course/Subject: AP Computer Science Grade Level: 11-12 # of Weeks: 4

Identify Desired Results

Common Core Standards and College Board Topic Outline Standards

- **II. Program Implementation**

The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

- B. Programming constructs
  - 4. Control
    - e. Understand and evaluate recursive methods

- **III. Program Analysis**

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

- B. Debugging
  - 1. Categorize errors: compile-time, run-time, logic.
  - 2. Identify and correct errors.
  - 3. Employ techniques such as using a debugger, adding extra output statements, or hand-tracing code.
- G. Analysis of algorithms
  - 1. Informal comparisons of running times
  - 2. Exact calculation of statement execution counts

- **V. Standard Algorithms**

Standard algorithms serve as examples of good solutions to standard problems. Many are intertwined with standard data structures. These algorithms provide examples for analysis of program efficiency.

- C. Sorting
  - 1. Selection
  - 2. Insertion
  - 3. Mergesort

Enduring Understandings Generalizations of desired understanding via essential questions (Students will understand that …)	**Essential Questions** Inquiry used to explore generalizations
• Recursion is the process by which a method calls on itself with modified data to solve a problem. • Everything that can be done recursively can also be done iteratively. • Recursion often simplifies certain algorithms that can be done iteratively.	• What is recursion? • Why is recursion necessary? • How is it recursion related to iteration? • What are some techniques that can be used to analyze recursive algorithms? • Why is it useful to sort lists? • How do Selection Sort, Insertion

• There are many classic algorithms which lend themselves to being implemented recursively (fibonacci, Towers of Hanoi, etc) • Analysis of recursive algorithms relies on identification of the base case and the recursive call. • Sorting of lists is a strategy that is used often to aid in efficient data search and retrieval. • There are different sorting algorithms which are useful under varying circumstances.	Sort and Merge sort manipulate the data within a list? • What is the most efficient sorting algorithm and how does the initial configuration of the items in the list affect this?

Expected Performances
What students should know and be able to do

Students will know the following:

- Recursion is a programming strategy by which a function invokes itself.
- The base case in a recursive algorithm controls when the recursion ends.
- The recursive call determines the conditions under which the recursion continues.
- The three sorting algorithms covered on the AP Computer Science exam are Selection Sort, Insertion Sort and Merge Sort.
- The basics of how the three sorting algorithms covered on the AP test work.

Students will be able to do the following:

- Identify the base case scenario in a recursive algorithm.
- Identify the recursive call in a recursive algorithm.
- Hand trace recursive algorithms to predict program behavior.
- Analyze the outcome of recursive programs correctly.
- Create programs that use recursive algorithms.
- Write code that swaps data within an array without data loss.
- Determine the Big O notation for the sorting algorithms expressed as a function of the number of items in the list to be sorted.
- Analyze sorting code to determine which algorithm is used.
- Show how a list changes at each step as the various sorting algorithms are run on them.

Character Attributes

- Perseverance

Technology Competencies

- Students collaborate with peers or others to solve problems and to develop solutions using technology tools and resources.
- Students identify and define object-oriented programming terminology.
- Students identify and explain programming structures.

Develop Teaching and Learning Plan

Teaching Strategies:	Learning Activities:
• Demonstration of coding concepts through analysis of worked examples. • Guided Practice and Demonstration • Vocabulary in context. • Use of Inception movie clip to demonstrate the concept of a base case in recursion. • Use of Russian nesting dolls to explain the concept of recursion. • Teacher shows videos that use folk dances to simulate the InsertionSort and SelectionSort algorithms. • Use of a deck of cards manipulative to explain MergeSort. • Flipped Learning (Students read material and complete an inductive assignment on the reading in order to come into class with questions, misconceptions, etc) • Discussion of questions /misconceptions re: content read. • Leveled assignments are offered for students who need remediation or more challenge. • Circulate and assist while students are completing assignments. • Assign certain advanced students to be “mentors” to struggling learners.	• Students complete a set of inductive learning Lab Exercises on the 3 lessons taught in the unit. • Students complete vocabulary worksheets for the 3 lessons taught in the unit. • Students answer AP style multiple choice questions. • Students answer AP style free response questions. • Students will complete Russian Doll simulation program that uses recursion to determine the number of dolls nested within a doll object and the name of the doll that occurs last in the alphabet within the set of nesting dolls. • Students will complete base case/recursive call identification worksheets. • Students will complete recursive algorithm analysis worksheets. • Students will complete a sorting algorithm worksheet that indicates the contents of the list after each pass of the different sorting algorithms. • In pairs, students will complete the Benchmarks program that runs the various sorting algorithms on lists of different sizes and outputs the algorithm run time in milliseconds. They will then plot the runtimes against the list sizes in Excel to determine how closely the curve plotted approximates the expected result. • Students pair/group to discuss and correct completed work. • For extra help and remediation, students watch videos based on content taught in class. • Students use given rubrics to self-evaluate programs.

Assessments	
Performance Task(s)	**Other Evidence**
Authentic application to evaluate student achievement of	Application that is functional in a classroom context to

desired results designed according to GRASPS (one per marking period)	evaluate student achievement of desired results
	• Quizzes (1 every two lessons) • Unit Test • Various coding-based Performance Assessments (detailed above) • Homework completion • AP-style Multiple Choice questions • AP-style Free-Response Questions

Suggested Resources

- Blue J IDE: www.bluej.org
- Bradley Kjell's Online Introduction to Java Course—Central Connecticut State University: http://chortle.ccsu.edu/CS151/cs151java.html
- Cook, Charles E. Blue Pelican Java. College Station: Virtualbookworm.com, 2005. Print.
- Litvin, Maria, and Gary Litvin. Java Methods: Object-oriented Programming and Data Structures. Andover, MA: Skylight Pub., 2011. Print.
- College Board Computer Science A AP Central Website: http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html
- Lew Tube Videos: http://www.thecubscientist.com/APCS/indexAPCS_HW.html
- Sorting Algorithms With Stacking Cups

http://comscigate.com/HW/cs201Deitel/sortingCups.htm

- Select-sort with Gypsy folk dance

http://www.youtube.com/watch?v=Ns4TPTC8whw&feature=endscreen&NR=1

- Insert-sort with Romanian folk dance http://www.youtube.com/watch?v=ROalU379l3U&feature=related
- College Board AP® Computer Science A Recursion Module Handbook
- Various Teacher-Created assignments

New Milford Public Schools

Committee Member(s): Shana Bergonzelli Unit Title: Miscellaneous and Test Prep	Course/Subject: AP Computer Science Grade Level: 11-12 # of Weeks: 5

Identify Desired Results

Common Core Standards and College Board Topic Outline Standards

- **III. Program Analysis**
  The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.
  - A. Testing
    - 1. Test classes and libraries in isolation.
    - 2. Identify boundary cases and generate appropriate test data.
    - 3. Perform integration testing.
  - B. Debugging
    - Categorize errors: compile-time, run-time, logic.
    - Identify and correct errors.
    - Employ techniques such as using a debugger, adding extra output statements,or hand-tracing code.
  - E. Understand error handling
    - Understand runtime exceptions.
  - F. Reason about programs
    - Pre- and post-conditions
    - Assertions
  - H. Numerical representations and limits
    - Representations of numbers in different bases

Enduring Understandings Generalizations of desired understanding via essential questions (Students will understand that …)	**Essential Questions** Inquiry used to explore generalizations
• Computers use the binary number system to transfer data at the most primitive level. • Two other useful numbering systems in technology are hexadecimal and octal. Numbers are often expressed in hexadecimal in computer code. • Java exceptions occur when there is a runtime problem in a program. The JRE stops execution and throws an exception to notify the user where the program went wrong. • There are various strategies that can be employed during the AP exam to save time and maximize	• Why are the different numbering systems useful? • What is the purpose of Java runtime exceptions? • What are some strategies I can employ that will aid me in achieving success on the AP exam?

results.	

Expected Performances
What students should know and be able to do

Students will know the following:

- The binary numbering system is base-2.
- The octal numbering system is base-8.
- The hexadecimal numbering system is base-16.
- The five exceptions tested on the AP Exam are NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, ClassCastException, and IllegalArgumentException.

Students will be able to do the following:

- Convert numbers between decimal, binary, hexadecimal and octal numbering systems by hand.
- Identify the conditions under which the exceptions listed above are thrown.
- Use strategies for answering questions on the AP exam efficiently and effectively.
- Self-identify knowledge gaps and weaknesses pertaining to the AP content.

Character Attributes

- Perseverance

Technology Competencies

- Students collaborate with peers or others to solve problems and to develop solutions using technology tools and resources.
- Students identify and define object-oriented programming terminology.
- Students identify and explain programming structures.

Develop Teaching and Learning Plan

Teaching Strategies:	Learning Activities:
• Demonstration of coding concepts through analysis of worked examples. • Guided Practice and Demonstration • Vocabulary in context. • Demonstrate the “quartet rule” for converting between binary and hexadecimal. • Guided practice on exceptions using code that demonstrates examples of the conditions under which the various exceptions are thrown. • Pass out list of Top 10 things to remember for the AP Exam (common kinds of questions and a prioritized topic list).	• Students complete online binary/octal/hexadecimal conversion questions and check their answers using a conversion calculator. • Students complete full practice exams under normal exam conditions. • Students self-score exams to using a diagnostic list to determine weaknesses. • Students implement free-response answers in the development environment to identify mistakes and debug code. • Students work to remediate weaknesses by completing packet of related questions (various packets on various topics will be available).

• Demonstrate various best practices for success on the exam (two-pass system, answer ALL questions, etc.). • After practice tests are administered, teacher reviews exam questions commonly missed by students.	

Assessments

Performance Task(s) Authentic application to evaluate student achievement of desired results designed according to GRASPS (one per marking period)	**Other Evidence** Application that is functional in a classroom context to evaluate student achievement of desired results
	• Various coding-based Performance Assessments (detailed above) • Homework completion • Practice Tests scored according to the College Board scale.

Suggested Resources

- Blue J IDE: www.bluej.org
- Cook, Charles E. Blue Pelican Java. College Station: Virtualbookworm.com, 2005. Print.
- Practice-It! AP Style Multiple Choice questions: http://practiceit.cs.washington.edu/
- CodingBat (was JavaBat) online problem solving website: http://codingbat.com/java
- College Board Computer Science A AP Central Website: http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html
- Teukolsky, Roselyn. *Barron's AP Computer Science A*. 6th ed. Hauppauge, NY: Barron's Educational Series, 2013. Print.
- Litvin, Maria. *Be Prepared for the AP Computer Science Exam in Java*. 5th ed. Andover, MA: Skylight Pub., 2013. Print.

New Milford Public Schools

Committee Member(s): Shana Bergonzelli Unit Title: Capstone Project	Course/Subject: AP Computer Science Grade Level: 11-12 # of Weeks: 5

Identify Desired Results

Common Core Standards and College Board Topic Outline Standards

- **III. Program Analysis**
  The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.
  - A. Testing
    - 1. Test classes and libraries in isolation.
    - 2. Identify boundary cases and generate appropriate test data.
    - 3. Perform integration testing.
  - B. Debugging
    - Categorize errors: compile-time, run-time, logic.
    - Identify and correct errors.
    - Employ techniques such as using a debugger, adding extra output statements,or hand-tracing code.
  - F. Reason about programs
    - Pre- and post-conditions
    - Assertions
- **VI. Computing in Context**

An awareness of the ethical and social implications of computing systems is necessary for the study of computer science. These topics need not be addressed in detail but should be considered throughout the course.

- A..System reliability
- B..Privacy
- C..Legal issues and intellectual property
- D. Social and ethical ramifications of computer use

Enduring Understandings Generalizations of desired understanding via essential questions (Students will understand that …)	**Essential Questions** Inquiry used to explore generalizations
• Programming is often a team effort. • Programmers often have different roles when collaborating on a large program. • Programming teams need to leverage team members' strengths effectively to solve problems. • Programmers need to use special software and tools when collaborating on a large program. • Planning is a crucial part of success when developing a large program in a team scenario.	• How do programmers collaborate to solve problems in the real world? • What tools do professional programmers need to assist them as they develop software? • What are some strategies programmers can use to ensure that they develop robust software?

Expected Performances
What students should know and be able to do

Students will know the following:

- Different software roles allow developers to leverage their strengths on a large team project and get more work done.
- Source control software helps developers work concurrently on a project on the same files and resolves conflicts when multiple changes are made to the same sections of code.
- UML diagrams are useful when planning a project. They help developers visualize the structure of a program and model relationships and source code dependencies among the files in a project.
- A robust test plan ensures that defects in the program are caught and corrected before the software is put in use.
- A bug database allows developers to keep track of defects in the code and whether they are a work on progress or fixed.

Students will be able to do the following:

- Design the main logic or “engine” for a program.
- Design the user interface for a program.
- Use source control software to collaborate on a large program.
- Use UML diagrams and flowcharts to plan a program.
- Independently learn and apply two programming concepts or structures that were not taught in class.

Character Attributes

- Perseverance
- Responsibility
- Cooperation

Technology Competencies

- Identify and select appropriate delivery methods and tools for digital media projects.

Develop Teaching and Learning Plan

Teaching Strategies:

- Explain how BlueJ utilizes UML diagrams and have students examine examples.
- Use presentation software to demonstrate various software engineering concepts.
- Demonstrate effective and ineffective user interfaces.
- Create mixed ability groups for project.
- Circulate and assist while students are completing projects.

Learning Activities:

- Students will compile a list of different programming job descriptions and explore career interests related to each role.
- Students will practice creating UML diagrams from existing programs by using the BlueJ IDE as an example.
- Students will research, download and use a software source control package to collaborate on the project.
- Students will each maintain a common working directory for their project that will allow the source control software to resolve conflicts and merge changes.
- Students will maintain a journal that

	details their project progress. • Students will create a detailed proposal for their project and get teacher approval. • Students will assume various roles while developing project (manager, architect, developer, tester, defect resolution) • Students will develop a flowchart and UML diagram for project. • Students will design a user interface for their project using a graphics program or pencil and paper. • Students will describe why user interface was designed the way it was and how it enhances the experience for the user and makes the program easy to interact with. • Students will develop a testing plan for their project and maintain a bug data base.

Assessments	
Performance Task(s) Authentic application to evaluate student achievement of desired results designed according to GRASPS (one per marking period)	**Other Evidence** Application that is functional in a classroom context to evaluate student achievement of desired results
Goal: Design, plan and code an original program from scratch that solves a problem or implements a game. Present the program to the class. Role: Programmer, Tester. Audience: Whole class, users of the program. Situation: In groups, students will propose a project that implements a game or solves a problem. Students will need to develop the program from scratch using concepts taught in class as well as two computer science concepts or structures which were not discussed in class. Students will need to use source control collaboration software to work on their project together. Students will	• Project Proposal. • Project Journal and notes • Evidence of source control software use. • Project flowchart. • Project UML diagram. • Project user interface design. • Project test plan and bug database.

demonstrate their program to the class along with a visual developed on presentation software. Product or Performance: Completed, running program. Presentation. Accompanying paper. Standards for Success: Graded by rubric.	

Suggested Resources

- Blue J IDE: www.bluej.org
- Github source control software: https://github.com/
- *MyFuture.com*. Unites States Department of Defense, n.d. Web. 06 Aug. 2015.
- *Stack Overflow*. Stack Exchange, Inc., n.d. Web. 06 Aug. 2015.
- *FOLDOC - Computing Dictionary*. N.p., n.d. Web. 06 Aug. 2015.
- Various teacher created resources.