College Board Curriculum Requirements

The Nexit AP Java course is fully College Board aligned and covers all seven curriculum requirements extensively as shown in the table below. The curriculum requirements laid out by the College Board are:

- ❖ CR1: Teaches students to design and implement computer-based solutions to problems.
- ❖ CR2a: Teaches students to use and implement commonly used algorithms.
- ❖ CR2b: Teaches students to use commonly used data structures.
- ❖ CR3: Teaches students to select appropriate algorithms and data structures to solve problems.
- ❖ CR4: Teaches students to code fluently in an object-oriented paradigm using the programming language Java.
- ❖ CR5: Teaches students to use elements of the standard Java library from the AP Java subset in Appendix A of the AP Computer Science A Course Description.
- ❖ CR6: Includes a structured-lab component composed of a minimum of 20 hours of hands-on lab experiences.
- ❖ CR7: Teaches students to recognize the ethical and social implications of computer use.

Course Overview and Goals

The Nexit AP Java course is a year-long course designed to help students master the basics of Java and equip them to successfully pass the College Board AP Computer Science A Exam at the end of the school year. All learning materials and resources teachers and students need for a successful year-long AP Java course can be found on the Nexit website.

Learning Environment: The course utilizes a blended classroom approach. The content is fully web-based, with students writing and running code in the browser. Teachers utilize tools and resources provided by Nexit to leverage time in the classroom and give focused 1-on-1 attention to students. Each unit of the course is broken down into lessons. Lessons consist of video tutorials, short quizzes, example programs to explore, and written programming exercises, adding up to over 100 hours of hands-on programming practice in total. [CR6] Several units have free response questions that have students consider the applications of programming and incorporate examples from their own lives.

Programming Environment: Students write and run Java programs in the browser using the Nexit editor. [CR1] [CR6]

Quizzes: At the end of each unit, students take a summative multiple choice unit quiz in the style of the AP Exam that assesses their knowledge of the Java concepts covered in the unit. Included in each lesson is a formative short multiple choice quiz.The course also provides an AP Test Practice unit with a cumulative AP Practice Multiple Choice Test and several Free Response questions.

Prerequisites

There are no official prerequisites for the Nexit AP Java course, however we recommend that students take our Introduction to Computer Science prior to AP Java (more info at Nexit.com/library). Students who have completed our Intro to CS course will be able to apply knowledge of concepts covered in the Intro course to the more advanced setting of the AP Java course. It is also expected that students know basic English and algebra. Students should be comfortable with functions and function notation, such as *f(x) = x + 2* and *f(x) = g(h(x))*.

Course Breakdown

Unit 1: Introduction to Programming in Java (3 weeks)

Objectives / Topics Covered [CR1]	● Commands ● Defining vs. Calling Methods ● Designing methods ● Program entry points ● Control flow ● Looping ● Conditionals ● Classes ● Commenting code ● Preconditions and Postconditions ● Top Down Design

Assignments / Labs [CR1] [CR6]	● 34 Karel Programming Exercises in total ● Program-specific tasks for Karel the Dog ○ Example Exercise: Maze Karel Karel is stuck in a maze. Help him escape and find the tennis ball at the end. Your job is to give commands to Karel to help navigate the maze and end up on the tennis ball. Karel should end up facing East. ● Teach Karel new commands like `turnRight()` or `makePancakes()` ○ Example Exercise: Pancakes Karel is the waiter. He needs to deliver a stack of pancakes to the guests on the 2nd, 4th, and 6th avenue. Each stack of pancakes should have three pancakes. Create a method called `makePancakes()` to help Karel solve this problem. ● Solve large Karel problems by breaking them down into smaller, more manageable problems using Top Down Design ○ Example Exercise: The Two Towers In this program, Karel should build two towers of tennis balls. Each tower should be 3 tennis balls high. At the end, Karel should end up on top of the second tower, facing East. ● Using control structures and conditionals to solve general problems ○ Example Exercise: Random Hurdles Write a program that has Karel run to the other side of first street, jumping over all of the hurdles. However, the hurdles can be in random locations. The world is fourteen avenues long. ○ Example Exercise: Super Cleanup Karel Karel's world is a complete mess. There are tennis balls all over the place, and you need to clean them up. Karel will start in the bottom left corner of the world facing east, and should clean up all of the tennis balls in the world. This program should be general enough to work on any size world with tennis balls in any locations.

Unit 2: Basic Java (9 weeks)

Objectives / Topics Covered [CR1] [CR5] [CR7]	• Printing • Variables • Types • Arithmetic Expressions • Casting `ints` and `doubles` • Input/Output • Errors • Loops • Conditionals • De Morgan's Laws • Short Circuit Evaluation • Debugging • Nested Control Structures • Working with the Java `String` class • Understand computer ethics such as acceptable use policies, copyright, intellectual property, and privacy
Assignments / Labs [CR1] [CR5] [CR6] [CR7]	• Several programming exercises to master each of the topics above. 1-3 exercises per topic for a total of 19 exercises. • Example Exercises ○ Add Fractions In this program you will ask the user for 4 integers that represent two fractions. First ask for the numerator of the first and then the denominator. Then ask for the numerator and denominator of the second. Your program should add the two fractions and print out the result. ○ Print the Odds Write a program that prints the odd numbers from 1 to 100. ○ Three Strings Write a program that asks the user for three strings. Then, print out whether the first string concatenated to the second string is equal to the third string. • To discuss computer ethics, prompt students to write a short positional argument about a real world issue connected to computer ethics, such as publishing software without properly debugging it or downloading a copyrighted program and giving it away for free.

Unit 3: Methods (3 weeks)

Objectives / Topics Covered [CR1] [CR5]	● Methods ● Parameters ● Return values ● Javadocs ● `@param` ● `@return` ● Understand how to iterate over a String and process each character ● Java Exceptions ● Compile-Time vs Run-Time Exceptions ● Java `String` class and methods ● Java `Character` class and methods 　○ Quick overview of static methods, more detail in next Unit
Assignments / Labs [CR1] [CR5] [CR6]	● Several programming exercises to master each of the topics above. 27 exercises in total ● Example Exercises: 　○ Parameter passing 　　■ Echo 　　Write a method called echo that takes one String parameter called `text` and one int parameter called `numTimes` and prints out that String that number of times. 　○ Return values 　　■ Average 　　Write a method called average that takes two doubles and returns a double that's the average of those two numbers. 　○ Javadocs 　　■ Is Divisible 　　Write a method that returns whether a is divisible by b. Provide proper Javadoc style comments above the method signature. 　　Your method signature should be 　　`public boolean isDivisible(int a, int b)` 　○ `String` class 　　■ First and Last 　　Write a method that returns a String that is just the first and last character of the given string. 　　Your return value should be only two characters long. You can assume that the given string will not be empty.

	The method signature should be `public String firstAndLast(String str)` ○ `Character` class ■ Is it an Integer? Given a string, determine if it is an integer. For example the string “123” is an integer, but the string “hello” is not. It is an integer if all of the characters in the string are digits. Return true if it is an integer, or false if it is not. Hint: There is a method `Character.isDigit()` that takes a char as an argument and returns a boolean value. ○ String processing ■ Replace Letter Write a method that replaces all instance of one letter with another. For example, `replaceLetter("hello", 'l', 'y')` returns `"heyyo"`

Unit 4: Classes and Object Oriented Programming (6 weeks)

Objectives / Topics Covered [CR1] [CR4] [CR5]	● Using classes as a client ● Classes vs Objects ● Class methods ● Instance variables ● Constructors ● Visibility ● Information hiding ● `this` ● `static` ● `super` ● The Java `Math` class and methods `(abs, pow, sqrt, sin, cos)` ● Creating random values with the Nexit `Randomizer`class ● Designing classes ● Creating classes ● Getter and setter methods ● Inheritance ● Method overloading ● Local variables and scope ● Comparing objects vs primitive types

	• Abstract classes • packages • Polymorphism • Interfaces • Modifying classes to implement interfaces • `Object` is the superclass of all classes

Assignments / Labs [CR1] [CR4] [CR5] [CR6]	• Several programming exercises to master each of the topics above. 35 exercises in total.
	• Examples
	  ○ Using the Student Class
	  In this program we have a `Student` class in Student.java and a tester program at StudentTester.java.
	  If you open up StudentTester.java you will see we have a bit of code there already. We've created two new students, Alan and Ada.
	  We create a `Student` instance by calling the constructor and passing in the first name, last name, and grade level as an integer.
	  Your task is to create a `Student` with your information! Once you have created the `Student`, print it out to the console.
	  ○ Design and implement a `Fraction` class from scratch, including a constructor, getter and setter methods, a `toString` method, and methods to add, subtract, and multiply by other `Fraction` objects.
	  ○ Implement a `RockPaperScissors` class with a `getWinner(String user, String computer)` method that allows a user to play the game Rock, Paper, Scissors against a computer that picks moves randomly.
	  ○ Add an abstract method to an existing `Shape` class called `public abstract double getPerimeter()` and implement this method on each of the Shape subclasses, `Square`, `Rectangle`, `Pentagon`, and `Circle`
	  ○ Fun with Solids
	  Given the `Solid` abstract class, extend it with:
	  `Pyramid`
	  `Cylinder`
	  `RectangularPrism`
	  `Sphere`
	  Make sure to create the constructor, `volume` and `surfaceArea` methods for each class (the `Math` class will come in handy).
	  Also extend `RectangularPrism` with `Cube`.

	○ Modify the `Fraction` class to implement the `Comparable` interface

Unit 5: Data Structures (6 weeks)

Objectives / Topics Covered [CR1] [CR2b] [CR3] [CR4] [CR5]	● Declaring and initializing arrays ● Constructing `ArrayLists` ● Indexing into arrays/`ArrayLists` ● Iterating over arrays/`ArrayLists` ● Getting the length of an array/`ArrayLists` ● `ArrayIndexOutOfBoundsException` ● `IndexOutOfBoundsException` ● Understand array variables are references to objects ● Arrays/`ArrayLists` as parameters and return values ● Inserting and deleting array/`ArrayList` elements ● Wrapper classes - `Double`, `Integer` ● Storing objects/primitives in arrays vs. `ArrayLists` ● Numerical representations of integers ○ Representations of non-negative integers in different bases ○ Implications of finite integer bounds ● The `List` interface ● Declaring and initializing 2-D rectangular arrays ● Using nested loops to iterate through 2-D arrays ● row-major order ● Students reminded about indices starting at 0 ● Constructing, adding to, and iterating through `HashMaps` ● Deciding which data structures to use when designing a class

Assignments / Labs
[CR1] [CR2b] [CR3] [CR4] [CR5] [CR6]
	● Several programming exercises to master each of the topics above. 23 exercises in total.
	● Examples
	  ○ Write a method that returns the index of the minimum value in an array
	  ○ Write a method that returns the first value in an `ArrayList`
	  ○ See how an `ArrayList` works under the hood. Write an `ExpandingArray` class that stores an array as an instance variable and supports the methods
	`public void add(int index, int element)`
	`public void add(int element)`
	`public int remove(int index)`
	`public int size()`
	`public String toString()`
	  ○ Write the method

	`public int sumRow(int[][] matrix, int row)` Which sums row row in the 2D array called matrix. ○ Explore and add to the code for a `BlackJack` game with a `Card` class, `Deck` class, `Hand` class, and `BlackJack` class ○ Implement the game Battleship with several incremental checkpoints ■ Implement the `Ship` class ■ Implement the `Location` class ■ Implement the `Grid` class ■ Implement adding a `Ship` to a `Grid` ■ Design and implement the `Player` class ■ Design and implement the `Battleship` class ■ Add extra features to the game

Unit 6: Algorithms and Recursion (3 weeks)

Objectives / Topics Covered [CR1] [CR2a] [CR3] [CR5]	● What is an algorithm? ● Algorithms in real life ● Implementing and using Sequential Search ● Implementing and using Binary Search ● Comparing relative run times of Sequential and Binary Search ● Brief introduction to Big-Oh ● Counting comparisons in searches and sorts ● Insertion Sort ● Selection Sort ● Merge Sort ● Pros and cons of each sorting algorithm ● Divide and Conquer ● Recursion ● `java.util.Arrays` ● Sorting and searching with both arrays and `ArrayLists`
Assignments / Labs [CR1] [CR2a] [CR3] [CR5] [CR6]	● Interactive visualization to explore how each sorting algorithm works ● Several exercises to master the topics above. 8 in total ● Example Exercises ○ Implement a sequential search function that takes an `ArrayList<Integer>` as a parameter ○ Modify the sequential search and binary search functions to return the number of comparisons made in each search, which one is more efficient? Test your functions on arrays of various sizes. ○ Modify insertion sort to sort elements in descending order

	○ Write a recursive function to reverse a string ○ Mergesort is a complicated algorithm, but how complicated is it? In this exercise, we'll be taking our example code from before and adding a cool feature: at every recursive step, print out to the console what the two halves are that are going to be merged together.

Unit 7: AP Test Practice (3 weeks)

Objectives / Topics Covered [CR1]	● Students know what to expect on the AP Exam ● Practice solving AP Exam type multiple choice questions ● Practice solving AP Exam type free response questions
Assignments / Labs [CR1] [CR6]	● Cumulative Final AP Review Multiple Choice Test with immediate feedback ● 4 multipart Free Response Questions with immediate feedback

Unit 8: Final Project (3 weeks)

Objectives / Topics Covered [CR1] [CR4]	● Allow students to think creatively about the applications of the concepts covered in the course ● Scoping a project ● Designing an application from scratch ● Incremental development ● Testing ● Debugging
Assignments / Labs [CR1] [CR4] [CR6]	● Brainstorm ideas for a final project ● Plan out milestones for incremental development ● Design the different classes you will create for this project ● Create your final product

Unit 9: Optional Supplemental Materials

Objectives / Topics Covered [CR1] [CR4] [CR7]	● Extra practice with AP CS A concepts ○ String processing ○ Recursion ○ Designing Classes ○ Arrays and `ArrayLists` ○ Searching and sorting algorithms ● File reading / writing ● The Java `Scanner` class

	• The Java `BufferedReader` and `BufferedWriter` classes • Running Java programs outside of the browser • Running Java programs from the command line • The Java `main` method • Computing in Context ○ Understand computer ethics such as acceptable use policies, copyright, intellectual property, privacy, and the implications of developing software used by real people in real life situations
Assignments / Labs [CR1] [CR4] [CR6] [CR7]	• Several additional exercises and advanced projects covering the topics listed above