Language Features and other Testable Topics

Tested in the AP CS A Exam	Notes	Not tested in the AP CS A Exam, but potentially relevant/useful
Comments `/* */`, `//`, and `/** */` Javadoc `@param` and `@return` comment tags		Javadoc tool
Primitive Types `int`, `double`, `boolean`		`char`, `byte`, `short`, `long`, `float`

Operators
Arithmetic: `+`, `-`, `*`, `/`, `%`
Increment/Decrement: `++`, `--`
Assignment: `=`, `+=`, `-=`, `*=`, `/=`, `%=`
Relational: `==`, `!=`, `<`, `<=`, `>`, `>=`
Logical: `!`, `&&`, `||`
Numeric casts: `(int)`, `(double)`
String concatenation: `+`
	1, 2, 3, 4, 5	`&`, `|`, `^`
		`(char)`, `(float)`
		`StringBuilder`
		Shift: `<<`, `>>`, `>>>`
		Bitwise: `~`, `&`, `|`, `^`
		Conditional: `?:`

Object Comparison object identity `(==, !=)` vs. object equality `(equals)`, `String compareTo`		implementation of `equals` `Comparable`
Escape Sequences `\"`, `\\`, `\n` inside strings		`\'`, `\t`, `\unnnn`
Input / Output `System.out.print`, `System.out.println`	6	`Scanner`, `System.in`, `System.out`, `System.err`, `Stream` input/output, GUI input/output, parsing input: `Integer.parseInt`, `Double.parseDouble` formatting output: `System.out.printf`

Tested in the AP CS A Exam	Notes	Not tested in the AP CS A Exam, but potentially relevant/useful

Exceptions
`ArithmeticException`,
`NullPointerException`,
`IndexOutOfBoundsException`,
`ArrayIndexOutOfBoundsException`,
`IllegalArgumentException`
		`try/catch/finally`
		`throw, throws`
		`assert`

Arrays 1-dimensional arrays, 2-dimensional rectangular arrays, initializer list: `{ ... }`, row-major order of 2-dimensional array elements	7, 8	`new` *type*`[]{ ... }`, ragged arrays (non-rectangular), arrays with 3 or more dimensions

Control Statements
`if, if/else,`
`while, for,`
enhanced `for` (for-each),
`return`
		`switch,`
		`break, continue,`
		`do-while`

Variables parameter variables, local variables, `private` instance variables: visibility `(private)` `static` (class) variables: visibility `(public, private)`, `final`		`final` parameter variables, `final` local variables, `final` instance variables

Methods
visibility `(public, private)`,
`static, non-static,`
method signatures,
overloading, overriding,
parameter passing
	9, 10	visibility `(protected)`,
		`public static void`
		`main(String[] args)`,
		command line arguments,
		variable number of parameters,
		`final`

Constructors `super(), super(`*args*`)`	11, 12	default initialization of instance variables, initialization blocks, `this(`*args*`)`

Tested in the AP CS A Exam	Notes	Not tested in the AP CS A Exam, but potentially relevant/useful
Classes `new`, visibility `(public)`, accessor methods, modifier (mutator) methods Design/create/modify class. Create subclass of a superclass `(abstract, non-abstract)`. Create class that implements an interface.	13, 14	`final`, visibility `(private, protected)`, nested classes, inner classes, enumerations
Interfaces Design/create/modify an interface.	13, 14	
Inheritance Understand inheritance hierarchies. Design/create/modify subclasses. Design/create/modify classes that implement interfaces.		
Packages `import` *packageName.className*		`import` *packageName*`.*` , `static` import, `package` *packageName* , class path
Miscellaneous OOP "is-a" and "has-a" relationships, `null`, `this`, `super.`*method(args)*	15, 16	`instanceof` *(class)* cast `this.`*var*, `this.`*method*`(`*args*`)`,
Standard Java Library `Object`, `Integer, Double`, `String`, `Math`, `List<E>, ArrayList<E>`	17, 18	`clone`, autoboxing, `Collection<E>`, `Arrays, Collections`

Notes

1. Students are expected to understand the operator precedence rules of the listed operators.
2. The increment/decrement operators `++` and `--` are part of the AP Java subset. These operators are used only for their side effect, not for their value. That is, the postfix form (for example, `x++`) is always used, and the operators are not used inside other expressions. For example, `arr[x++]` is not used.
3. Students need to understand the "short circuit" evaluation of the `&&` and `||` operators.
4. Students are expected to understand "truncation towards `0`" behavior as well as the fact that positive floating-point numbers can be rounded to the nearest integer as

   `(int)(x + 0.5)`, negative numbers as `(int)(x - 0.5)`.
5. String concatenation `+` is part of the AP Java subset. Students are expected to know that concatenation converts numbers to strings and invokes `toString` on objects.
6. User input is not included in the AP Java subset. There are many possible ways for supplying user input: e.g., by reading from a `Scanner`, reading from a stream (such as a file or a URL), or from a dialog box. There are advantages and disadvantages to the various approaches. The exam does not prescribe any one approach. Instead, if reading input is necessary, it will be indicated in a way similar to the following:

   ```
   double x = /* call to a method that reads a floating-point number */;
   ```

   or

   ```
   double x = ...;    // read user input
   ```
7. Both arrays of primitive types (e.g., `int[]`, `int[][]`) and arrays of objects (e.g., `Student[]`, `Student[][]`) are in the subset.
8. Students need to understand that 2-dimensional arrays are stored as arrays of arrays. For the purposes of the AP CS A Exam, students should assume that 2-dimensional arrays are rectangular (not ragged) and the elements are indexed in row-major order. For example, given the declaration

   ```
   int[][] m = {{1, 2, 3}, {4, 5, 6}};
   ```

   `m.length` is 2 (the number of rows), `m[0].length` is 3 (the number of columns), `m[r][c]` represents the element at row `r` and column `c`, and `m[r]` represents row `r` (e.g., `m[0]` is of type `int[]` and references the array `{1, 2, 3}`).

   Students are expected to be able to access a row of a 2-dimensional array, assign it to a 1-dimensional array reference, pass it as a parameter, and use loops (including for-each) to traverse the rows. However, students are not expected to analyze or implement code that replaces an entire row in a 2-dimensional array, such as

   ```
   int[][] m = {{1, 2, 3}, {4, 5, 6}};
   int[] a = {7, 8, 9};
   m[0] = a;    // Outside the Subset
   ```

9. The `main` method and command-line arguments are not included in the subset. In free-response questions, students are not expected to invoke programs. In the *AP Computer Science Labs*, program invocation with `main` may occur, but the `main` method will be kept very simple.
10. Students are required to understand when the use of `static` methods is appropriate. In the exam, `static` methods are always invoked through a class (explicitly or implicitly), never an object (i.e., *ClassName.staticMethod*() or *staticMethod*(), not *obj.staticMethod*()).
11. If a subclass constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass.
12. Students are expected to implement constructors that initialize all instance variables. Class constants are initialized with an initializer:

    ```
    public static final int MAX_SCORE = 5;
    ```

    The rules for default initialization (with `0`, `false` or `null`) are not included in the subset. Initializing instance variables with an initializer is not included in the subset. Initialization blocks are not included in the subset.
13. Students are expected to write interfaces or class declarations when given a general description of the interface or class.
14. Students are expected to extend classes and implement interfaces. Students are also expected to have knowledge of inheritance that includes understanding the concepts of method overriding and polymorphism. Students are expected to implement their own subclasses.

    Students are expected to read the definition of an abstract class and understand that the abstract methods need to be implemented in a subclass. Students are similarly expected to read the definition of an interface and understand that the abstract methods need to be implemented in an implementing class.
15. Students are expected to understand that conversion from a subclass reference to a superclass reference is legal and does not require a cast. Class casts (generally from `Object` to another class) are not included in the AP Java subset. Array type compatibility and casts between array types are not included in the subset.
16. The use of `this` is restricted to passing the implicit parameter in its entirety to another method (e.g., *obj.method*`(this)`) and to descriptions such as "the implicit parameter `this`". Students are not required to know the idiom "`this.`*var* = *var*", where *var* is both the name of an instance variable and a parameter variable.
17. The use of generic collection classes and interfaces is in the AP Java subset, but students need not implement generic classes or methods.
18. Students are expected to know a subset of the constants and methods of the listed Standard Java Library classes and interfaces. Those constants and methods are enumerated in the Java Quick Reference (Appendix B).

APPENDIX B

Exam Appendix — Java Quick Reference

Accessible methods from the Java library that may be included on the exam

class java.lang.Object

- `boolean equals(Object other)`
- `String toString()`

class java.lang.Integer

- `Integer(int value)`
- `int intValue()`
- `Integer.MIN_VALUE` // minimum value represented by an `int` or `Integer`
- `Integer.MAX_VALUE` // maximum value represented by an `int` or `Integer`

class java.lang.Double

- `Double(double value)`
- `double doubleValue()`

class java.lang.String

- `int length()`
- `String substring(int from, int to)` // returns the substring beginning at `from`
  // and ending at `to-1`
- `String substring(int from)` // returns `substring(from, length())`
- `int indexOf(String str)` // returns the index of the first occurrence of `str`;
  // returns `-1` if not found
- `int compareTo(String other)` // returns a value < 0 if `this` is less than `other`
  // returns a value = 0 if `this` is equal to `other`
  // returns a value > 0 if `this` is greater than `other`

class java.lang.Math

- `static int abs(int x)`
- `static double abs(double x)`
- `static double pow(double base, double exponent)`
- `static double sqrt(double x)`
- `static double random()` // returns a `double` in the range [0.0, 1.0)

interface java.util.List<E>

- `int size()`
- `boolean add(E obj)` // appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` // inserts `obj` at position `index` ($0 \le \texttt{index} \le \texttt{size}$),
  // moving elements at position `index` and higher
  // to the right (adds 1 to their indices) and adjusts size
- `E get(int index)`
- `E set(int index, E obj)` // replaces the element at position `index` with `obj`
  // returns the element formerly at the specified position
- `E remove(int index)` // removes element from position `index`, moving elements
  // at position `index + 1` and higher to the left
  // (subtracts 1 from their indices) and adjusts size
  // returns the element formerly at the specified position

class java.util.ArrayList<E> implements java.util.List<E>