Review of AP Computer Science A

Table of Contents:

|  |  |
| --- | --- |
| **Unit** | **Concepts** |
| [Unit 0: Java Basics](#Unit_0_Content)  [Problem Set](#Unit_0_Problems) | Data types, variables, static void methods |
| Unit 1: Definite Loops  Problem Set | For loops, nested for loops |
| Unit 2: Parameters and Returns  Problem Set | Methods with parameters, method overloading, return types |
| Unit 3: Program Logic  Problem Set | If/else if/else, while, nested conditional statements, input validation |
| Unit 4: Objects  Problem Set | Constructors, non-static methods, this keyword, accessors and mutators, equals, toString, null, has-a |
| [Unit 5: Arrays/ArrayLists](#Unit_5_Content)  [Problem Set](#Unit_5_Problems) | Accessing all elements, length, get/set/add/remove/size |
| Unit 6: Inheritance  Problem Set | Interfaces, abstract classes, super and sub classes, super(), overriding, Object, is-a |
| *Unit 7: Searching and Sorting*  *Problem Set (Missing)* | *Insertion vs. Selection vs. Mergesort and Sequential vs. Binary Search, Relative computational efficiency (O(1) vs. O(n) vs. O(n^2) vs. O(n\*log n))* |
| Unit 8: Recursion  Problem Set | Recursive tracing |

Note: This is does not contain everything we have learned this semester but it is a good survey of the topics.

# Unit 0: Java Basics

## General Basics

All java programs need a class that matches the name of the .java file exactly. The syntax for a class is:

public class MyClass {

//methods here

}

All java programs then need a main method to be runnable. The syntax for a main method is:

public static void main(String[] args) {

//statements here

}

When a program is run, the computer begins with the first line of main and processes each line sequentially from top to bottom.

## Output

Printing to the console can be done with System.out.print(“text”) and System.out.println(“text”).

### print() vs. println()

print leaves the cursor on the same line after the statement is executed

println moves the cursor to the next line after the statement is executed

For example:

System.out.print(“This ”);

System.out.println(“text ”);

System.out.println(“will ”);

System.out.print(“appear “);

System.out.print(“on three “);

System.out.println(“lines.”);

Would output:

This text

will

appear on three lines.

## Data Types

### Primitives

|  |  |
| --- | --- |
| int | 15, -7, 0, Integer.MIN\_VALUE = -2147483648, Integer.MAX\_VALUE = 2147483647 |
| double | 6.3, -2.718, 0.0, Math.PI |
| boolean | true, false |
| char | ‘a’, ‘z’, ‘3’, ‘?’ |

### Non-Primitive Data Types:

|  |  |
| --- | --- |
| String | “foo”, “A string is really a list of characters” |

## Variables

### Syntax

Declaration: type name;

Assignment: name = value;

Declaration and assignment: type name = value;

Examples: int x = 3, String city = “Seattle”, boolean isHungry = true

### Operations

Java uses PMDAS from left to right to determine order of operations when evaluating an expression.

#### Ints

Defined operations between int and int are: +, -, /, \*, %

+, -, \* all behave as expected

/ floors the answer to the nearest integer (e.g. 7 / 2 = 3, 9 / 10 = 0)

% (called mod) computes the remainder after division (5 % 2 = 1, 11 % 7 = 4, 32 % 8 = 0)

#### Doubles

Defined operations between double and double or double and int are: +, -, /, \*

+, -, /, \* all behave as expected, except occasionally Java has rounding issues

Note: When operations between ints and doubles are performed, ints are automatically promoted to doubles and the result is a double.

#### Strings

Defined operation between String and String or String and other is: +

+ concatenates a String with other to create a longer String. The result of String + other is always a String.

Examples: “foo” + “bar” = “foobar”, “house” + 12 = “house12”, “TRUE” + false = “TRUEfalse”

### Casting

A data type may be converted to another by casting it like so:

(type)expr

For example, (int)3.4 would be equivalent to the int 3 and (double)-2 would be equal to -2.0.

Note: Since multiplying an int by a double automatically promotes the int we could also have done (1.0 \* -2).

### Shorthand Operations

|  |  |
| --- | --- |
| x += 1; | x = x + 1; |
| x -= 2; | x = x – 2; |
| x \*= 3; | x = x \* 3; |
| x /= 4; | x = x / 4; |
| x %= 5; | x = x % 5; |
| x++; | x += 1; or x = x + 1; |
| x--; | x -= 1; or x = x – 1; |

## Static methods

A static method is a way to structure code to be easily reused.

### Method Declaration

Syntax:

public static void methodName() {

//statements here

}

### Calling Methods

To call a method you write the name of a method followed by empty parentheses.

methodName();

Statements written in a method are not executed unless the method is called in main or another method.

When a method is called, the computer will execute all of the statements of the called method in order from top to bottom. When the method’s closing curly brace is hit, the computer will then go to the next line after the method call. For example if the following program were run:

|  |  |
| --- | --- |
| public static void main(String[] args) {  System.out.println(“Starting main”);  methodA();  methodB();  System.out.println(“Done with main”);  }  public static void methodA() {  System.out.println(“Starting A”);  System.out.println(“Done with A”);  }  public static void methodB() {  System.out.println(“Starting B”);  methodA();  System.out.println(“Done with B”);  } | Output:  Starting main  Starting A  Done with A  Starting B  Starting A  Done with A  Done with B  Done with main |

# Unit 0: Java Basics – Problems

1. What are the results of the following expressions?
2. 3 + 2 – 4 \* 3
3. 813 % 100 / 3 + 2.4
4. 177 % 100 % 10 / 2
5. 2 + 19 % 5 - (11 \* (5 / 2))
6. "1" + 2 + 3 + "4" + 5 \* 6 + "7" + (8 + 9)
7. Which expression will determine the 1’s digit for an integer, x?

[A] x / 10

[B] x % 1

[C] x / 1

[D] x % 10

1. What are the values of a, b, and c after the following code is executed?

|  |  |
| --- | --- |
| int a = 5;  int b = 10;  int c = b;  a = a + 1;  b = b – 1;  c = c + a; | Result:  a =  b =  c = |

1. What is stored in x after the following code is executed?

int x = 5;

int y = x / 2;

double z = x \* 2.5 + y;

x = (int)z / y;

x =

1. What is stored in x after the following code is executed?

int x = 15;

int y = 4;

x /= y;

x \*= y;

x =

1. Suppose the following methods are declared.

public static void a() {

System.out.println(“a”);

}

public static void b() {

System.out.println(“b”);

a();

}

public static void c() {

a();

b();

System.out.println(“c”);

}

What is the output if the following statements are executed?

|  |  |
| --- | --- |
| Calls:  public static void main(String[] args) {  a();  b();  c();  } | Output: |

Unit 1: Definite Loops

Loops are helpful for iteration, such as iterating through arrays or collections of objects. There are 3 types of loops that you may see on the AP exam.

Loop Syntax

For Loops:

for(initialization; termination condition; update statement){

Statements //body of loop

}

While Loops:

while(Boolean test){

Statements //body of loop

}

For-each Loops are specifically used to iterate over an array or collection:

for(SomeType element: collection){

Statements //body of loop

}

Nested Loop Syntax

Nested For Loops:

for(first initialization; first termination condition; first update statement){

for(second initialization; second termination condition; second update statement){

Statements //body of loop

}

}

**Please note**:

1. Brackets are optional, but without them, only one of code in the statement body is executed.
2. It’s possible for a for loop or a while loop to never be executed. A for loop is never executed if its termination condition is met before the loop is entered, and a while loop will never run if its Boolean test is false before the loop is entered.

Examples:

|  |  |
| --- | --- |
| for(int i=0; i<5; i++){  System.out.print(i+” ”);  } | 0 1 2 3 4 |
| int i = 10  while(i>0){  System.out.print(i+” ”);  i-=2;  } | 10 8 6 4 2 |
| int[] arr = {3, 7, 15};  for(int number: arr){  System.out.print(number+” ”);  } | 3 7 15 |
| for(int i=1; i<5; i++){  for(int j=1; j<3; j++){  System.out.print(i\*j+” ”);  }  System.out.println;  } | 1 2 3  2 4 6  3 6 9  4 8 12 |

Unit 1: Definite Loops - Problems

1. Consider the following code segment

int value = 15;

while(value < 28){

System.out.println(value);

value++;

}

What are the first and last numbers output by the code segment?

First Last

1. 15 27
2. 15 28
3. 16 27
4. 16 28
5. 16 29

Question 2 and 3 refer to the following code segment

int k = *a random number such that* 1 <= k <= n

for (int p = 2; p <= k; p++)

for(int r = 1; r < k; r++)

System.out.println(“Oh hey there”);

2. What is the minimum number of times that “Oh hey there” will be printed?

1. 0
2. 1
3. 2
4. n-1
5. n-2

1. What is the maximum number of times that “Oh hey there” will be printed.
2. 2
3. n-1
4. n-2
5. (n-1)2
6. n2

1. Consider the following code segment

int num1 = 0;

int num2 = 3;

while((num2 != 0) && ((num1 / num2) >= 0){

num1 = num1 + num2;

num2 = num2 - 1;

}

What are the values of num1 and num2 after the while loop completed its execution?

1. num1 = 0, num2 = 3
2. num1 = 8, num2 = -1
3. num1 = 4, num2 = 1
4. num1 = 6, num2 = 0
5. The loop will never complete its execution because a division by zero will generate an ArithmeticException
6. Consider the following two static methods, where f2 is intended to be the iterative version of f1.

public static int f1(int n) {

   if(n < 0){

return 0;

}else{

Return (f1(n-1) + n\*10);

}

}

public static int f2(int n) {

int answer = 0;

while(n > 0){

answer = answer + n\*10;

n--;

}

return answer;

}

The method f2 will always produce the same results as f1 under which of the following conditions?

1. n < 0
2. n = 0
3. n > 0
4. I only
5. II only
6. III only
7. II and III only
8. I, II, and III

1. Consider the following output:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4 5 6

Which of the following code segments will produce the output shown above?

1. for (int j = 1; j <= 6; j++){

for(int k = 1; k < j; k++)

System.out.print(“ ”+k);

System.out.println();

}

1. for (int j = 1; j <= 6; j++){

for(int k = 1; k <= j; k++)

System.out.print(“ ”+j);

System.out.println();

}

1. for (int j = 1; j <= 6; j++){

for(int k = 1; k <= j; k++)

System.out.print(“ ”+k);

System.out.println();

}

1. for (int j = 1; j < 6; j++){

for(int k = 1; k <= j; k++)

System.out.print(“ ”+k);

System.out.println();

}

1. for (int j = 1; j < 6; j++){

for(int k = 1; k < j; k++)

System.out.print(“ ”+k);

System.out.println();

}

Unit 2: Parameters and Returns

Parameters and Returns

Some methods require a parameter or multiple parameters to run​. When the method is called, the value is stored into the parameter variable and the method's code executes using that value.​

If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.​ **Overloading** is when more than one method in a single class has the same name but different parameters.​ The main purpose of overloading is for code to be easier to read or use.​

Parameters can be changed while in a method, but those changes only live within that method. If you want to know the parameter value if it was changed in the method, then you must **return** a value.

Parameter Syntax

Writing a method with one parameter and calling a method with one parameter:

public static void method1 ( type name ) {​

    statement(s);​

}​

method1 (expression);​

Writing a method with multiple parameters and calling a method with multiple parameters:

public static void method2 ( type1 name1, type2 name2, … ,typeN nameN) {​

    statement(s);​

}​

method2 (expression1, expression2, …, expressionN);​

Common Errors

1. If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.​
2. The value passed to a method must be of the correct type.​
3. When primitive variables (int, double) are passed as parameters, their values are copied. Modifying the parameter will not affect the variable passed in.​

Examples

|  |  |
| --- | --- |
| public static void main(String[] args) {​      greeting(“hello”);​      greeting(“good day!”);​  }​  public static void greeting(String greet){  System.out.println(“The greeting is ”+ greet);  } | The greeting is hello​  The greeting is good day!​ |
| public static void main(String[] args) {​      nameAndAge(“Polly”, 3);​      nameAndAge(“Roger”, 456);​  }​  public static void nameAndAge(String name, int age){  System.out.println(“My name is ”+ name+” and my age is ”+age);  } | My name is Polly and my age is 3  My name is Roger and my age is 456 |
| public static void main(String[] args) {​      greeting();​​  }​  public static void greeting(String greet){  System.out.println(“The greeting is ”+ greet);  } | This is an error |
| public static void main(String[] args) {​     temperature (“Fifty-five degrees”);​     temperature (“55”);​  }​  public static void temperature(String temp){  System.out.println(“The temperature is ”+ temp);  }  public static void temperature(int temp){  System.out.println(“The temperature is ”+ temp);  } | The temperature is Fifty-five degrees  The temperature is 55 |

​​

Math Class

The Math Class is useful for common mathematical calculations. Note that the methods do not print to the console. Instead, each method produces ("returns") a numeric result.

The Math Class can be called through the following syntax:

Math.methodName(parameters)​

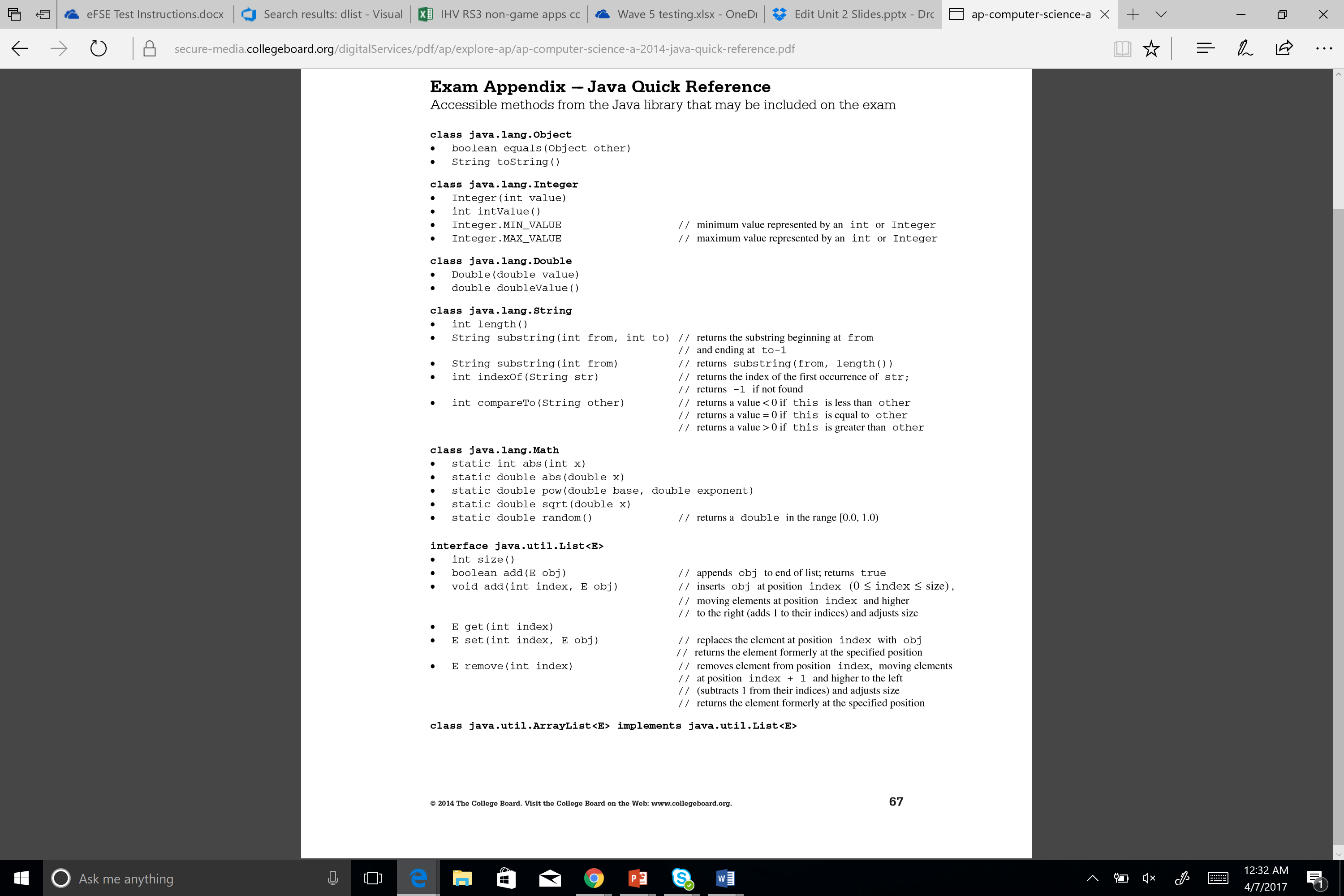
​

Example:​

​double squareRoot = Math.sqrt(121.0);​

System.out.println(squareRoot);           // 11.0​

These are the methods they will give you on the exam:



Strings

A Java string is a sequence of characters. They are objects of type String. Strings are Immutable, which means they cannot be changed after they’ve been created; the default constructor creates an empty string.​

Syntax

​String name = "text";​

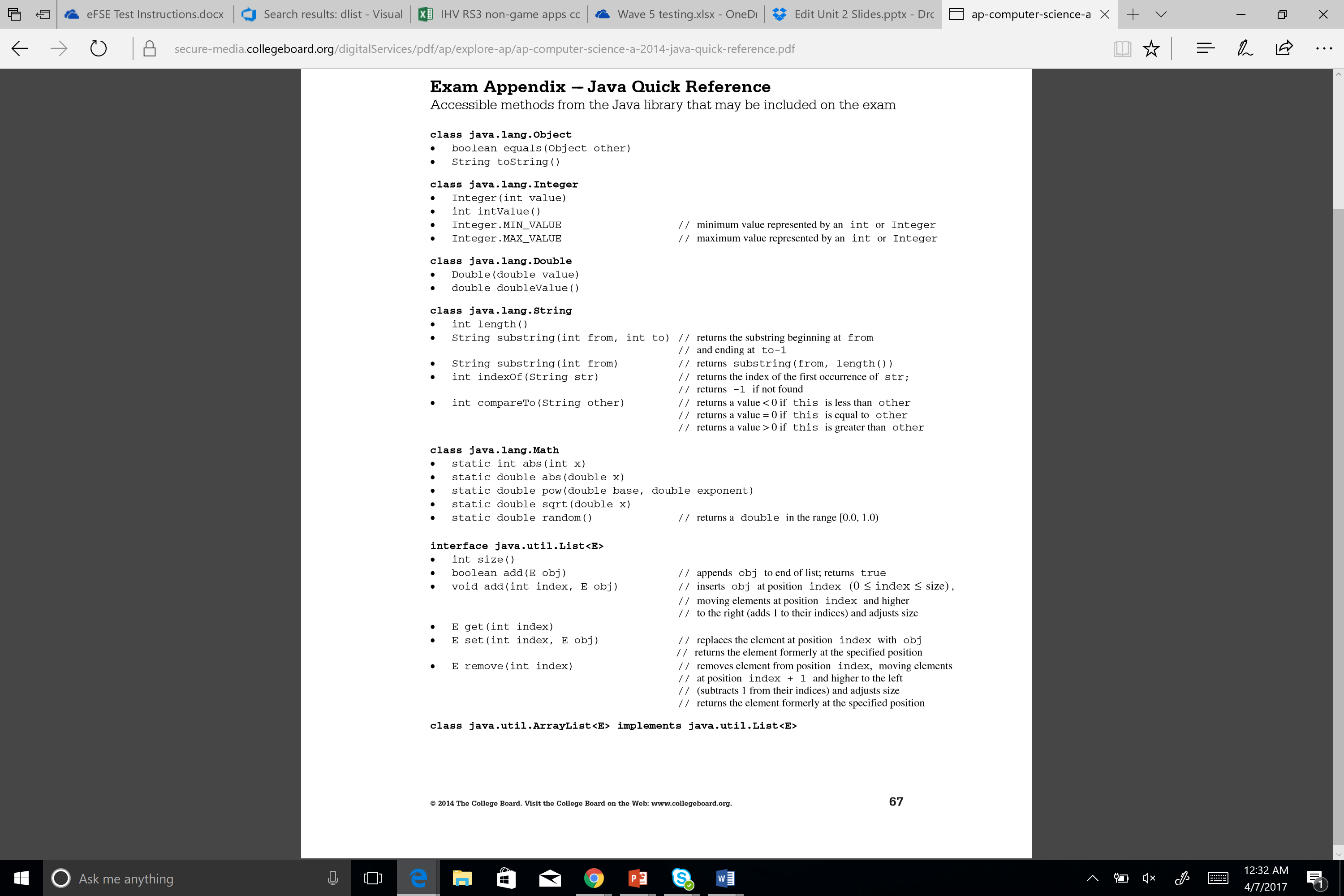
String name = expression;​

Examples:​

​String name = "Polly Roger";​

​String s = new String();

These are the methods they will give you on the exam:



Unit 2: Parameters and Returns - Problems

1. Consider the following code segment.

String str = “abcdef”;

for(int rep = 0; rep < str.length()-1; rep++){

System.out.print(str.substring(rep, rep+2));

}

What is printed as a result of executing this code segment?

1. abcdef
2. aabbccddeeff
3. abbccddeef
4. abcbcdcdedef
5. Nothing is printed because an IndexOutOfBoundsException is thrown

1. Assume that the following variable declarations have been made.

double d = Math.random();

double r;

Which of the following assigns a value to r from the uniform distribution over the range 0.5 <= 0 < 5.5?

1. r = d+0.5;
2. r = d+0.5\*5.0;
3. r = d\*5.0;
4. r = d\*5.0+0.5;
5. r = d\*5.5;

1. Consider the following method.

public String goAgain(String str, int index){

if(index >= str.length())

return str;

return str.goAgain(str.substring(index), index+1);

}

What is printed as a result of executing the following statement?

System.out.println(goAgain(“today”, 1));

(A) today

(B) todayto

(C) todayoday

(D) todayodayay

(E)todayodaydayayy

1. Which statement about parameters is false?
2. The scope of parameters is the method in which they are defined
3. Static methods have no implicit parameter this
4. Two overloaded methods in the same class must have parameters with different names
5. All parameters in Java are passed by value
6. Two different constructors in a given class can have the same number of parmeters
7. Consider these declarations:

String s1 = “crab”;

String s2 = new String(“crab”);

String s3 = s1;

Which expression involving these strings evaluates to true?

I s1 == s2

II s1.equals(s2)

III s3.equals(s2)

1. I only
2. II only
3. II and III only
4. I and II only
5. I, II, and III

6. Consider this method:

public static String doSomething(String s) {

   final String BLANK = “ ”; //String with one space

String str = “”; //empty string

String temp;

for(int i = 0; i<s.length(); i++){

temp = s.substring(i, i+1);

if(!temp.equals(BLANK))

str+=temp;

}

Return str;

}

Which of the following is the most precise description of what doSomething does?

1. It returns s with all its blanks removed
2. It returns s unchanged
3. It returns a String that is equivalent to s with all its blanks removed
4. It returns a String that is an exact copy of s
5. It returns a String that contains s.length() blanks

# Unit 3: Program Logic

## If statement

If statements are used to control the flow of the program. Executes a block of statements only if a test is true.

### Syntax

if (test) { // test is an expression or Boolean value

<statement>;

<statement>;

}

## If/else statement

### Syntax

if (test) {

<statement>;

<statement>;

…

} else {

<statement>;

<statement>;

…

}

## If/else if/else statement

### Syntax

if (test) {

<statement>;

<statement>;

…

} else if{

<statement>;

<statement>;

…

} else {

<statement>;

<statement>;

…

}

## Boolean Expressions

if statements and for loops both use logical tests implemented with Boolean expresions

a == b // evaluates to true if a equals b

a != b // evaluates to true if a does not equal b

a < b // evaluates to true if a is less than b

a > b // evaluates to true if a is greater than b

a <= b // evaluates to true if a is less than or equal to b

a >= b // evaluates to true if a is greater than or equal to b

## Logical Operators

Tests can be combined using logical Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Result** |
| && | and | (2 == 3) && (-1 < 5) | false |
| || | or | (2 == 3) || (-1 < 5) | true |
| ! | not | !(2 == 3) | true |

|  |  |  |  |
| --- | --- | --- | --- |
| **p** | **q** | **p && q** | **p || q** |
| true | true | true | true |
| true | false | false | true |
| false | true | false | true |
| false | false | false | false |

## String Comparison Method

Strings are Objects so we need to use the methods provided to compare them. Using the relational operators doesn’t evaluate the value of the string but rather the string object(the object reference)

Objects are compared using a method named equals.

|  |  |
| --- | --- |
| **Method** | **Description** |
| equals(**str**) | whether two strings contain the same characters |
| equalsIgnoreCase(**str**) | whether two strings contain the same characters, ignoring upper vs. lower case |
| startsWith(**str**) | whether one contains other's characters at start |
| endsWith(**str**) | whether one contains other's characters at end |
| contains(**str**) | whether the given string is found within this one |
| CompareTo(Object o) | Compares the string to the Object parameter. Instead of returning a Boolean the method returns the value 0 if the argument is a string lexicographically equal to this string; returns a value less than 0 if the argument is a string lexicographically greater than this string; and returns a value greater than 0 if the argument is a string lexicographically less than this string. |

## While loops

### Syntax

while (test) {

statement(s);

}

### Example

int num = 1; // initialization

while (num <= 200) { // test

System.out.print(num + " ");

num = num \* 2; // update

}

// output: 1 2 4 8 16 32 64 128

### Definite loop

Executes a known number of times.

### Indefinite loop

One where the number of times its body repeats is not known in advance.

### Sentinel values

A value that signals the end of user input.

sentinel loop: Repeats until a sentinel value is seen.

## Do While Loop

Performs its test at the *end* of each repetition.

Guarantees that the loop's {} body will run at least once.

do {

**statement(s)**;

} while (**test**);

# Unit 3: Program Logic - Problems

1. How many times does the following while loop execute the body of the loop:

int x = 1;

while(x < 100) {

System.out.print(x + “ “);

X += 10;

}

1. Given the following method: ​

public static void mystery(int x) {

int y = 1;

int z = 0;

while (2 \* y <= x) {

y = y \* 2;

z++;

}

System.out.println(y + " " + z);

}

What is the output of the call mystery(74);

1. Translate the following statements into logical tests:

z is odd

z is not greater then y’s square root

y is positive

either x or y is even

1. Consider the following method.

​public static void ifElseMystery1(int x, int y) {

int z = 4;

if (z <= x) {

z = x + 1;

} else {

z = z + 9;

}

if (z <= y) {

y++;

}

System.out.println(z + " " + y);

}

What output is produced with the following calls

ifElseMystery(3, 20);

ifElseMystery(5,5);

ifElseMystery(6,10);

# Unit 4: Objects

## Definition

A **Java** **object** is a combination of data and procedures working on the available data. An **object** has a state and behavior. The state of an **object** is stored in fields (variables), while methods (functions) display the **object**'s behavior.

## Scanner, String

Scanner and String are both Objects. The data in String contains the value(all the letters/characters in the string). String has many useful behaviors like .indexOf(char c).

## State and Behavior

State is the collection of data that make up an object.

Behavior is the collection of things an object can do

## “blueprint”

When you instantiate an object using the new keyword, an instance of that object is created based on the data and behaviors defined on that object.

## Object vs Clients

Clients have main methods and can operate by themselves. Objects don’t have main methods. Clients use objects like tools.

# Unit 4 : Objects – Questions

1. What is the state and behavior of a String object?

* The state is its sequence of characters, and the behavior is its methods, such as length and indexOf.
* The state is methods that methods that return values, such as length or charAt, and the behavior is methods that change the string, such as toUpperCase or substring.
* The state is the memory occupied by the object, and the behavior is what you do with that memory, such as storing characters in it.
* The state is the string itself, and the behavior is what you do with it, such as putting it in an array or list.
* The state is the location the text came from, such as a file or from the Scanner, and the behavior is when you stave that string to a new location, such as to a new file.

1. What is the definition of an object?
2. What are instance variables?
3. What are constructors used for? How many constructors can an object have?
4. Describe encapsulation and why it is important for objects to be encapsulated.

# Unit 5: Arrays and ArrayLists

## Arrays

### Syntax

Declaration: type[] name;

Initialization: name = new type[size];

Declaration and Initialization: type[] name = new type[size];

type[] name = {item, item, …, item};

When you initialize an array with new, it constructs an array with space for size items of the given type. Then, all of the elements will be initialized to zero or the equivalent of zero.

Examples:

|  |  |
| --- | --- |
| int[] a = new int[3]; | 0  0  0  a |
| double[] a = new double[3]; | 0.0  0.0  0.0  a |
| boolean[] a = new boolean[3]; | false  false  false  a |
| String[] a = new String[3]; | null  null  null  a |

#### A note about null

null is a reference to an object that doesn’t exist.

null cannot be dereferenced through . notation.

For example,

String a = null; //ok

System.out.println(a.length()); //throws a NullPointerException at runtime

Since a is referencing null (no object) it has no length, so the program complains when it is run.

### Array Accessing

An *index* is a number between 0 and the length of the array – 1 (inclusive) that is used to access a particular element of an array.

Accessing: arr[index]

Assignment: arr[index] = value;

Size: arr.length

The first element of an array is always arr[0] and the last element of an array is always arr[arr.length - 1].

Warning! Any attempt to access an element that is not a valid index will cause an ArrayOutOfBoundsException.

For example:

arr[arr.length] //error since the last element is at arr.length - 1.

|  |  |
| --- | --- |
| double[] temps = new double[3] ;  temps[0] = 13.5 ;  temps[1] = -3.14;  System.out.println(temps[0]);  System.out.println(temps[1]);  System.out.println(temps[2]); | Output:  13.5  -3.14  0.0 |

### For Loops

It is common to want to loop over all the elements in an array, either to print the array out, find the largest value, or determine if a value is contained in the array. That makes arrays and for loops the best of friends.

Process an array from front-to-back:

for (int i = 0; i < arr.length; i++) {

//do something with arr[i]

}

Process an array from back-to-front:

for (int i = arr.length - 1; i >= 0; i--) {

//do something with arr[i]

}

## 2D Arrays

### Syntax

Declaration: type[][] name;

Initialization: name = new type[int][int];

Declaration and Initialization: type[][] name = new type[int][int]; or

type[][] name = {{val,…},{val,…},…,{val,…}};

Number of rows: name.length

Number of columns: name[0].length

### Accessing 2D arrays

It is easiest to think of a 2D array as a matrix of rows and columns, where the first number is the row number and the second number is the column number. For example:

int[][] matrix = new int[3][3];

matrix[0][0] = 4;

matrix[1][1] = 5;

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 4 | 0 | 0 |
| 1 | 0 | 5 | 0 |
| 2 | 3 | 0 | 0 |

matrix[2][0] = 3

Could be thought of as the grid:

### 2D Arrays as Arrays of Arrays

It is important to know that 2D arrays are in fact arrays of arrays. For example:

matrix[0] points to the row array with elements {4, 0, 0}

matrix[1] points to the row array with elements {0, 5, 0} and

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 4 | 2 | 0 |
| 1 | 0 | 5 | 0 |
| 2 | 3 | 0 | 0 |

matrix[2] points to the row array with elements {3, 0, 0}.

Therefore if the following code were executed, matrix would look like this:

int[] a = matrix[0];

a[1] = 2;

### Nested For-Loops

To iterate over (look at) all of the elements of a 2D array, we typically use nested for loops.

Processes the 2D array in row-major order (row by row)

for (int row = 0; row < matrix.length; row++) {

for (int col = 0; col < matrix[0].length; col++) {

//do something with matrix[row][col]

}

}

Processes the 2D array in column-major order (column by column)

for (int col = 0; col < matrix[0].length; col++) {

for (int row = 0; row < matrix.length; row++) {

//do something with matrix[row][col]

}

}

## ArrayLists

ArrayLists differ from Arrays in that they are dynamically resizeable. When arrays are initialized, their size must be explicitly given. ArrayLists can be initialized empty and then filled with the add method.

Note about the <> usage. <Type> specifies the Class of the object that the ArrayList will contain.

### Syntax

Declaration: ArrayList<Type> name;

Initialization: name = new ArrayList<Type>();

Declaration and Initialization: ArrayList<Type> name = new ArrayList<Type>(); or

List<Type> name = new ArrayList<Type>();

### Useful Methods

|  |  |  |
| --- | --- | --- |
| ArrayList | Array | Function |
| Type list.get(int index) | arr[index] | Returns the object at index |
| void list.set(int index, Type val) | arr[index] = val | Sets the object at index to val |
| int list.size() | arr.length | Returns the number of elements in list |
| void list.add(Type val) | N/A | Adds the object val to the end of the list, size is incremented |
| void list.add(int index, Type val) | N/A | Inserts the object val at index, all objects to the right of val have their indexes shifted up one and size is incremented |
| Type list.remove(int index) | N/A | Removes and returns the object at index, all objects to the right of index have their indexes shifted down one and size is decremented |

Warning! Modifying an ArrayList while looping over the elements can cause unwanted behavior.

For example:

Intent: Remove all even integers.

for (int i = 0; i < list.size(); i++) {

if (list.get(i) % 2 == 0) {

list.remove(i);

}

}

Problem: If two adjacent integers are even, then the second integer will be skipped.

Solution: Process list from back-to-front or decrement i whenever remove is called.

|  |  |
| --- | --- |
| int start = list.size() - 1;  int end = 0;  for (int i = start; i >= end; i--)  {  if (list.get(i) % 2 == 0) {  list.remove(i);  }  } | int start = 0;  for (int i = start; i < list.size() - 1; i++)  {  if (list.get(i) % 2 == 0) {  list.remove(i);  i--;  }  } |

### Enhanced For Loops

To loop over every item in a List or array, you can use an enhanced for loop.

|  |  |  |
| --- | --- | --- |
| for (type name : list) {  //do something with name } | for (int i = 0; i < list.size(); i++) {  //do something with list.get(i)  } | This accesses each item in list but doesn’t allow you to modify the item (unless it is an object that you call a mutator method on). |

Enhanced for loops are read-only for all primitives.

# Unit 5: Arrays and ArrayLists - Problems

1. Write a static method called minIndex that returns the first index of the smallest element in an array of integers.

For example:

minIndex(new int[]{3, -1, 2, 4, -1}) would return 1.

public static int minIndex(int[] a) {

}

1. Write a static method that returns an int[] array of the first n multiples of 3 or 5.

For example:

threesAndFive(7) would return [3, 5, 6, 9, 10, 12, 15]

public static int[] threesAndFives(int n) {

}

1. Write a static method called swapPairs that takes an array of Strings as a parameter and swaps each pair of Strings. If the array of Strings has an odd number of Strings, then the last String is not moved.

For example:

String[] fruits = {“apple”, “pear”, “banana”, “mango”, “grape”};

swapPairs(fruits); //fruits is now {“pear”, “apple”, “mango”, “banana”, “grape”}

public static void swapPairs(String[] s) {

}

1. Consider the following method.

public ArrayList<Integer> mystery(int n) {

ArrayList<Integer> seq = new ArrayList<Integer>();

for (int k = 1; k <= n; k++) {

seq.add(new Integer(k \* k + 3));

}

return seq;

}

Which of the following is printed as a result of executing the following statement?

System.out.println(mystery(6));

1. [3, 4, 7, 12, 19, 28]
2. [3, 4, 7, 12, 19, 28, 39]
3. [4, 7, 12, 19, 28, 39]
4. [39, 28, 19, 12, 7, 4]
5. [39, 28, 19, 12, 7, 4, 3]
6. Consider the following method that is intended to return the sum of the elements in the array key.

public static int sumArray(int[] key) {

int sum = 0;

for (int i = 1; i <= key.length; i++) {

/\* missing code \*/

}

return sum;

}

Which of the following statements should be used to replace /\* missing code \*/ so that sumArray will work as intended?

1. sum = key[i];
2. sum += key[i -1];
3. sum += key[i];
4. sum += sum + key[i - 1];
5. sum += sum + key[i];
6. Consider the following method.

public static void arrayMethod(int[] nums) {

int j = 0;

int k = nums.length - 1;

while (j < k) {

int x = nums[j];

nums[j] = nums[k];

nums[k] = x;

j++;

k--;

}

}

Which of the following describes what the method arrayMethod does to the array nums?

1. The array nums is unchanged.
2. The first value in nums is copied to every location in the array.
3. The last value in nums is copied to every location in the array.
4. The method generates an ArrayIndexOutOfBoundsException.
5. The contents of the array nums are reversed.
6. Consider the following code segment.

int[] oldArray = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int[][] newArray = new int[3][3];

int row = 0;

int col = 0;

for (int value : oldArray) {

newArray[row][col] = value;

row++;

if ((row % 3) == 0) {

col++;

row = 0;

}

}

System.out.println(newArray[0][2]);

What is printed as a result of executing the code segment?

1. 3
2. 4
3. 5
4. 7
5. 8
6. Write a method called removeDuplicates that takes an ArrayList of Strings as a parameter and removes all duplicates from the list.

For example: Calling removeDuplicates on the list [“fuji”, “fuji”, “cameo”, “braeburn", “cameo”, “fuji”]

would turn the list into [“fuji”, “cameo”, “braeburn"].

public static void removeDuplicates(ArrayList<String> list) {

}

Unit 6: Inheritance

Inheritance is way to form new classes based on existing classes by taking on their attributes/behavior, grouping related classes​, and sharing code between two or more classes​. It is useful for ensuring that code isn’t duplicated, and making code easier to develop.

​

​One class can extend another by absorbing its data and/or behavior:

**superclass**: The parent class that is being extended​

**subclass**: The child class that extends the superclass and inherits its behavior and gets a copy of every field and method from superclass​

Syntax:

public class name extends superclass {…}​

​**is-a relationship**: A hierarchical connection where one category can be treated as a specialized version of another, such as every marketer is an employee​ and every legal secretary is a secretary​

**inheritance hierarchy**: A set of classes connected by is-a relationships that can share common code.​

**override**: To write a new version of a method in a subclass that replaces the superclass's version.​

No special syntax required to override a superclass method, just write a new version of it in the subclass.​ If the code for overriding includes a call to the superclass method, this can be done by using the super keyword.

**Polymorphism**: is the mechanism of selecting the appropriate methods for a particular object in a class hierarchy; in java this happen during runtime.

**Interface**: is a collection of related methods, either abstract (headers only) or default (implementation provided in the interface). Interfaces are implemented with the implements keyword. Interfaces cannot be instantiated, only their implementing classes.

**Abstract Class:** is a class that has either abstract or implemented methods. It can have a constructor, but an abstract class itself cannot be instantiated with the keyword new.

​

Unit 6: Inheritance - Problems

1. Consider the following class definitions

public class Data{

private int x;

public void setX(int n){ x= n;}

//other methods not shown

}

public class EnhancedData extends Data{

private int y;

public void setY(int n){ y=n; }

//other methods not shown

}

Assume the following declaration appears in a client program:

EnhancedData item = new EnhancedData( );

Which of the following statements would be valid?

I item.y = 16;

II item.setY(16);

III item.setX(25);

1. I
2. II
3. I and II
4. II and III
5. I, II, and III

2. When designing classes, which of the following would be the best reason to use inheritance?

(A) Inheritance allows you to write applications that require fewer base and super classes

(B) Inheritance allows the creation of a subclass that can use the methods of its superclass without rewriting the cod for those methods.

(C) Inheritance allows for data encapsulation, while noninherited classes do not allow for this

(D) Inheritance reduces the number of polymorphic structures encapsulated in applications

(E)Inheritance guarantees that the applications will compile and execute much more quickly

3. A bear is an animal and a zoo contains many animals, including bears. Three classes Animal, Bear, and Zoo are declared to represent animal, bear, and zoo objects. Which of the following is the most appropriate set of declarations?

(A)

public class Animal extends Bear{ … }

public class Zoo{

private Animal[] myAnimals;

…

}

(B)

public class Bear extends Animal{...}

public class Zoo{

private Animal[] myAnimals;

…

}

(C)

public class Animal extends Zoo

{

private Bear myBear;

…

}

(D)

public class Bear extends Animal, Zoo

{

…

}

(E)

public class Bear extends Animal implements Zoo

{

…

}

1. Consider the following class definitions.

Public class A

{

private int a1;

public void methodA( )

{

methodB( ); // Statement I

}

}

public class B extends A

{

public void methodB( )

{

methodA( ); //Statement II

a1=0; //Statement III

}

}

}

Which of the labeled statements in the methods shown above will cause a compile-time error?

1. I only
2. III only
3. I and II
4. I and III
5. II and III
6. Which of the following is (are) true of an interface?

I An interface can contain a constructor

II An interface can be instantiated

III All methods in an interface are abstract

1. I only
2. II only
3. III only
4. I and II only
5. I, II, and III

1. Consider the following three declarations.

I Integer obj1 = new Integer(7);

II Comparable obj 2 = new Integer(7);

III Comparable obj3 = new Comparable(7);

Which of these declarations is (are legal)?

1. I only
2. I and II only
3. I and III only
4. II and III only
5. I, II, and III
6. When designing a class hierarchy, which of the following should be true of a superclass?
7. A superclass should contain the data and functionality that are common to all the subclasses that inherit from the superclass
8. A superclass should be the largest, most complex class from which all other subclasses are derived
9. A superclass should contain the data and functionality that are only required for the most complex class
10. A superclass should have public data in order to provide access for the entire class hierarchy
11. A superclass should contain the most specific details of the class hierarchy
12. Consider the following class declarations.

public class Base{

private int myVal;

public Base(){

myVal = 0;

}

public Base(int x){

myVal = x;

}

}

public class Sub extends Base{

public Sub(){

super(0);

}

}

Which of the following statements will NOT compile?

1. Base b1 = new Base();
2. Base b2 = new Base(5);
3. Base s1 = new Sub();
4. Sub s2 = new Sub();
5. Sub s3 = new Sub(5);

Unit 8: Recursion

Introduction

A recursive method is a method that calls itself. The structure of all recursive methods is that there is a base case (this can be do nothing) and a recursive case separated in an if/else structure. The recursive case calls the method with different parameters that must ultimately call the recursive base case.

Typical Form of Recursion:

public static void recursive(int n) {

if (n == base\_case) { //base case

//process base case

} else { //recursive case

//recursively call recursive

}

}

Common Errors

Recursive calls don’t progress towards base case

A recursive method that does not progress towards a base case will cause infinite recursion (similar to a loop whose condition is always true). This is a programming logic error to avoid.

The following two code chunks are equivalent.

|  |  |
| --- | --- |
| Infinite Recursion | Infinite Loop |
| public static int recur(int n) {  if (n == 0) {  return 0;  } else {  return n + recur(n + 1);  }  recur(10); | int n = 0;  for (int i = 10; i >= 0; i++) {  n = n + i;  } |

Notice that each recursive call increases n but our base case is when n == 0. Similarly, the for loop condition is true while i >= 0 and since i is increasing at each step it will never end (until i equals Integer.MAX\_VALUE).

No base case given

Forgetting a base case can cause infinite recursion as well.

|  |  |
| --- | --- |
| Infinite Recursion | Infinite Loop |
| public static int recur(int n) {  return n + recur(n - 1);  }  recur(10); | int n = 0;  for (int i = 10; true; i--) {  n = n + i;  } |

There is no base case, so the program will continue calling itself forever. This is similar to the infinite loop where the condition is always true.

Tracing Recursive Code

To understand what a recursive method will do, it is helpful to make a table of all of the recursive calls. For example,

public static void funnyPrinter(String s) {

|  |  |  |
| --- | --- | --- |
| s | action | output |
| “cat” | print(“c”), funnyPrinter(“at”), print(“c”) | “cattac” |
| “at” | print(“a”), funnyPrinter(“t”), print(“a”) | “atta” |
| “t” | print(“t”), funnyPrinter(“”), print(“t”) | “tt” |
| “” | Do nothing |  |

if (s.length() > 0) {

System.out.print(s.substring(0,1));

funnyPrinter(s.substring(1));

System.out.print(s.substring(0,1));

}

}

funnyPrinter(“cat”);

Unit 8: Recursion – Problems

1. Write a recursive method that adds and returns the numbers from 1 to n inclusive.

public static int sum(int n) {

}

1. Write a recursive method that returns a String in reverse order. For example, reverse(“cow”) would return “woc”.

public static String reverse(String s) {

}

1. The following method is designed to determine whether a String is a palindrome. A palindrome is a String that reads the same forwards and backwards. For example, racecar, madam and poop are palindromes.

public static boolean isPalindrome(String s) {

if (s.length() == 0 || s.length() == 1) {

return true;

}

return s.substring(0,1).equals(s.substring(s.length() – 1)) &&

/\* missing code \*/;

}

What should replace /\* missing code \*/ so that isPalindrome works as intended?

1. isPalindrome(s.substring(1))
2. isPalindrome(s.substring(0,1))
3. isPalindrome(s.substring(1, s.length()))
4. isPalindrome(s.substring(1, s.length() – 1))
5. isPalindrome(s.substring(0,1) + s.substring(s.length() – 1))
6. Suppose the following method is declared.

public static int doubleTrouble(int n) {

if (n == 0) {

return 1;

} else {

return doubleTrouble(n – 1) + doubleTrouble(n – 1);

}

|  |  |
| --- | --- |
| n | result |
|  |  |

}

What will the call doubleTrouble(5) return?

What does doubleTrouble compute?

Is this an efficient algorithm? Why or why not?

1. Suppose the following method is declared.

public static int mystery(int a, int b) {

if (b == 0) {

return a;

} else {

return mystery(b, a % b);

}

}

What will the following calls return?

mystery(10, 5);

mystery(6, 9);

mystery(24, 18);

mystery(48, 56);

1. Consider the implementation of mergesort below. Assume that merge has been written correctly.

public static void mergesort(int[] a) {

if (a.length > 1) {

int[] left = Arrays.copyOfRange(a, 0, a.length/2);

int[] right = Arrays.copyOfRange(a, a.length/2, a.length);

/\* missing code \*/

merge(a, left, right);

}

}

What should replace missing code so that mergesort works as intended?