Java source code -> Java compiler -> Java byte code -> Java Virtual Machine -> machine code

c++ source code -> assembly code -> object code -> machine code

Java: portable, oop

c++: higher performance

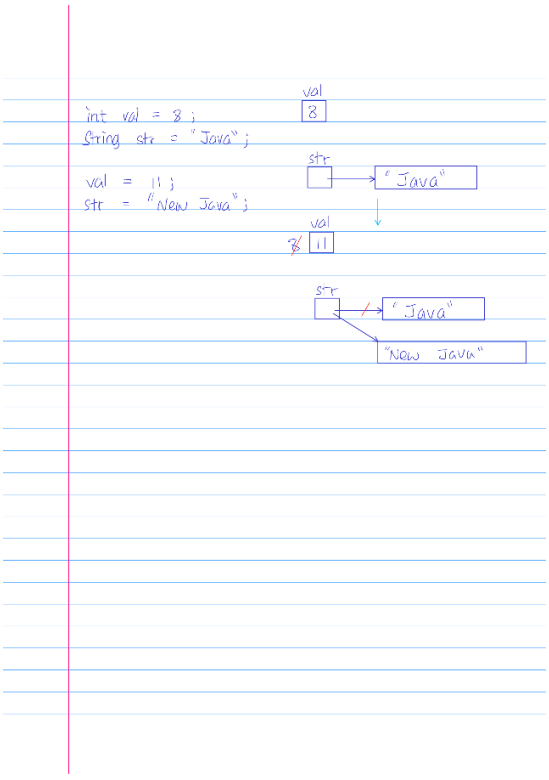
1. Generate a random number between 5 and 20

```
import java.util.Random;
Random rand = new Random();
int val = rand.nextInt(16) + 5;
// val between 5 and (5 + 16) – 1 = 20 inclusive
```

2. Differences between primitive type and reference type, draw diagrams

Primitive type variables hold the value of the type.

Reference type variables hold the memory address of the type.



3. Determine output for a segment of code

```
for (int i = 1; i <= n; i++)
```

```
{
        for (int j = i; j > 0; j--)
                System.out.print("#");
        System.out.println();
}
```

Output:

If n = 5,

i = 1, j = 1, #

i = 2, j = 2, ##

i = 3, j = 3, ###

I = 4, j = 4, ####

I = 5, j = 5, #####

4. Translate pseudocode to Java code

a. Creating an object of type X

```
X x = new X();
```

b. Invoke method m1 of this object sending an int value

```
x.m1(5);
```

c. Print the object

```
System.out.println(x);
```

5. What is a constructor and how do you set one up?

A constructor is a special method that is used to initialize objects. A constructor always has the same name as the class. A class can have multiple constructors where each has different number of parameters and different data type of parameters.

6. Set up a method that accepts some parameters and performs a task – determine if n is prime or power of 2, reverse an array

```
static boolean isPrime(int n) {
        for (int i = 2; i <= n / 2; ++i) {
                if (n % i == 0)
                        return false;
```

```
        }
        return true;
}
static boolean isPowOfTwo(int n) {
        while (n % 2 == 0) {
                n /= 2;
        }
        return n == 1;
}
static boolean isPowOfTwo(int n) {
        return (n & (n – 1)) == 0;
}
static void reverseArray(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        int tmp;
        while (start < end) {
                tmp = arr[start];
                arr[start] = arr[end];
                arr[end] = tmp;
                ++start;
                --end;
        }
}
```

7. Perform some work with a string using various string operations – given a string, convert the first letter of each word to uppercase; given a string, count vowels

```
static String firstLetterToUpper (String str) {
        String newStr = “”;
        String word;
```

```
        Scanner scan = new Scanner(str);
        scan.useDelimiter(" ");
        while (scan.hasNext()) {
                word = scan.next();
                word = word.subString(0, 1).toUpperCase() + word.subString(1);
                newStr += word + " ";
        }
        scan.close();
        return newStr.trim();
}
static int countVowels(Stirng str) {
        int count = 0;
        str = str.toLowerCase();
        for (int i = 0; i < str.length(); ++i) {
                if (str.charAt(i) == 'a' || str.charAt(i) == 'e' || str.charAt(i) == 'i' || str.charAt(i) == 'o' ||
str.charAt(i) == 'u')
                        ++count;
        }
        return count;
}
```

8. Complete the code for a class – constructors, getters, setters, equals, toString

```
public class Student {
        private String name;
        private String id;
        private double gpa;
        public Student(String name, String id, double gpa) {
                this.name = name;
                this.id = id;
                this.gpa = gpa;
```

```
        }
        public void setID(String name) {
                this.id = id;
        }
        public String getID() {
                return id;
        }
        public bool equals(Object other) {
                return (other instanceof Student) ? (id == ((Student) other).getID()) : false;
        }
        public String toString() {
                return “Name: “ + name + “\nStudent ID: “ + id + “\nGPA: “ + gpa;
        }
}
```

9. What are some levels of testing from earliest to latest?

Unit testing: the process of testing an individual software component.

Integration testing: the process of testing software components that are made up of other interacting components. Stresses the communication between components rather than the functionality of individual components.

System testing: the process of testing the software application as a whole rather than between software components.

Acceptance testing: demonstrate or run different test cases in front of customers or management.

Regression testing: running previous test cases after a change is made to a program to help ensure that the change did not introduce new errors.

Test cases: a description of the input and corresponding expected output of a code unit being tested.

10. Best loop to implement a sentinel loop, a counting loop, and a y/n loop?

while and do-while loop are best to implement a sentinel loop or y/n loop.

for loop is best to implement a counting loop.

11. Testing vs. debugging; black box vs. white box testing; when should you stop testing?

Testing attempts to ensure that the program will solve the intended problem under all the constraints specified in the requirements.

Debugging is the process of determining the cause of a problem and fixing it.

Black box testing is a software testing done by people without having knowledge of internal logic or code structure. Black box testing is focused on input and expected output of the software application.

White box testing is a software testing done by developers. Developers focus on the internal logic and code structure. In white box testing, code is visible to the testers.

12. Perform a simple if-else with the conditional operator – min of two values

```
min = (x < y) ? x : y;
```

13. Use a nested loop to generate a pattern like a rectangle or a triangle

```
for (int i = 0; i < row; ++i) {
        for (int j = 0; j < col; ++j) {
                System.out.print("$");
        }
        System.out.println();
}
for (int i = 0; i < row; ++i) {
        for (int j = 0; j < i + 1; ++j) {
                System.out.print("$");
        }
        System.out.println();
}
```

14. Draw a simple UML class diagram showing relationships between some classes

Movies uses DVDCollection.

DVDCollection has DVD;

15. Use the Coin or Die class to perform a simulation; roll two dice 100 times and count the number of times they have the same value

```
Die die1 = new Die();
Die die2 = new Die();
int count = 0;
```

```
for (int i = 0; i < 100; ++i) {
        if (die1.roll() == die2.roll())
                ++count;
}
```

16. Use an array to keep track of the counts for n items

```
Die die1 = new Die();
Die die2 = new Die();
int[] count = new int[11];
for (int i = 0; i < 100; ++i) {
        ++count[die1.roll() + die2.roll() – 2];
}
```

17. Describe three different relationships between classes

Dependency: RollDie use a Die.

Composition: PairOfDice has-a two Die.

Inheritance: LoadedDie is-a Die.

18. How is method overloading different from method overriding?

Method overloading: multiple methods share the same name with different number of parameters and/or different data type of parameters.

Method overriding: create a new method in a subclass that replaces the method with the same name in the superclass.

19. How try-catch works; checked exceptions vs. unchecked exceptions; what happens when an exception is thrown?

Checked exceptions: if a method could throw a checked exception, the program must be able to catch the exception.

Unchecked exceptions: if a method could throw an unchecked exception, the program is not required to catch the exception.

20. Set up a simple recursive method to print a string in reverse order; trace a recursive method such as factorial or summing values from 1 to n

```
public static printStringInReverse(String str) {
        if (str.length() < 1)
                return;
```

```
        System.out.print(str.charAt(str.length() - 1));
        printStringInReverse(str.substring(0, str.length() - 1));
}
```

```
public static int factorial ( int n ) {
    return (n <= 1) ? 1 : n * factorial(n-1);
}

factorial(4) = 4 * 6 = 24
    → factorial(3) = 3 * 2 = 6
        → factorial(2) = 2 * 1 = 2
            → factorial(1) = 1

public static int sum ( int n ) {
    return (n <= 1) ? n : n + sum(n-1);
}

sum(4) = 4 + 6 = 10
    → sum(3) = 3 + 3 = 6
        → sum(2) = 2 + 1 = 3
            → sum(1) = 1
```

21. Explain polymorphism and how to achieve polymorphism in Java

Polymorphism means an object can have many forms. Polymorphism is the ability of a reference variable to refer to objects of various types at different times. For example, polymorphism allows the method of the object type to be invoked instead of the reference type.