Scheme of Evaluation
Internal Assessment Test 1 – May 2022

Sub:	Programming in Java							**Code:**	18CS653
Date**:**	13/5/2022	Duration:	90mins	Max Marks:	50	**Sem:**	VI	**Branch:**	EEE/MECH

Note: Answer Any five full questions.

Question #		**Description**	**Marks Distribution**		**Max Marks**
1	a)	Explain breifly primtive type of data types in Java. 1. Integer 2. Byte 3. Short 4. Long 5. Char 6. Boolean 7. Float 8. Double	1M*7	7M	10M
1	b)	Explain the logical operator in Java with suitable example. AND OR NOT	1M*3	3M	
2	a)	Write a Java program to print following pattern. 0 1 2 3 4 5 6 7 8 9	5M	5M	10M
2	b)	Write a program to find factorial of a number 5. Input Loop output	1M 3M 1M	5M	

3	a)	How Java arrays are initialized and declared , explain with an example Array-syntax Array initialization example	1M 2M 2M	5M	10M
3	b)	Write a Java program to check entered number is prime or not. Input Check prime number output	1M 3M 1M	5M	
4	a)	Explain why java is a strongly typed language.		4M	10M
4	b)	Explain steps to execute simple java program and role of JVM in execution. Java code-helloworld Javac hello.java-- compilation Java hello - execution output	3M 1M 1M 1M	6M	
5	a)	Explain three OOP principles. i)Inheritance ii)Polymorphism iii)Encapsulation	2M*3	6M	10M
5	b)	Differentiate between narrowing and widening with proper example. Narrowing with example Widening with example	2M 2M	4M	
6	a)	Explain following with syntax and example. i)for-each ii) for loop For-each loop syntax with example For loop syntax with example	3M 3M	6M	10M
6	b)	Write a Java program to find the average of the array elements	1M 2M	4M	

		Input Code Output	1M		
7	a)	Explain the following operators in Java a)>>> b)<< c)? d) % e) &=	2M*5	10M	10M

Q. 1 a) Explain breifly primtive type of data types in Java.

Type	Size (bits)
byte	8
short	16
int	32
long	64
float	32
double	64
char	16
boolean	1

2.1. *int*

The first primitive data type we're going to cover is *int*. Also known as an integer, *int* type holds a wide range of non-fractional number values.

```
int x = 424_242; int y;
```

2.2. *byte*

byte is a primitive data type similar to *int*, except **it only takes up 8 bits of memory.** This is why we call it a byte. Because the memory size is so small, *byte* can only hold the values from -128 (-2^7) to 127 ($2^7 - 1$).

Here's how we can create *byte*:

```
byte b = 100;

byte empty;
```

2.3. *short*

The next stop on our list of primitive data types in Java is *short*.

If we want to save memory and *byte* is too small, we can use the type halfway between *byte* and *int*: *short*.

```
short s = 20_020; short s;
```

2.4. *long*

Our last primitive data type related to integers is *long*.

long is the big brother of *int*. **It's stored in 64 bits of memory**, so it can hold a significantly larger set of possible values.

```
long l = 1_234_567_890; long l;
```

2.5. *float*

We represent basic fractional numbers in Java using the *float* type. This is a single-precision decimal number. This means that if we get past six decimal points, the number becomes less precise and more of an estimate.

```
float f = 3.145f; float f;
```

2.6. *double*

It's stored in 64 bits of memory. **This means it represents a much larger range of possible numbers than *float*.**

```
double d = 3.13457599923384753929348D; double d;
```

2.7. *boolean*

The simplest primitive data type is *boolean*. It can contain only two values: *true* or *false*. **It stores its value in a single bit.**

```
boolean b = true; boolean b;
```

2.8. *char*

The final primitive data type to look at is *char*.

Also called a character, *char* is a 16-bit integer representing a Unicode-encoded character. Its range is from 0 to 65,535

```
char c = 'a'; char c = 65; char c;
```

Q. 2 b) Explain the logical operator in Java with suitable example.

Types of Logical (Boolean) Operators in Java

In Java, there are three types of logical operators. We have listed them in the below table.

Table: Logical Operators

Operators	Meaning
1. &&	AND operator
2. ||	OR operator
3. !	NOT operator

```
// Demonstrate the boolean logical operators.
class BoolLogic {
  public static void main(String args[]) {
    boolean a = true;
    boolean b = false;
    boolean c = a | b;
    boolean d = a & b;
    boolean e = a ^ b;
    boolean f = (!a & b) | (a & !b);
    boolean g = !a;
    System.out.println("        a = " + a);
    System.out.println("        b = " + b);
    System.out.println("      a|b = " + c);
    System.out.println("      a&b = " + d);
    System.out.println("      a^b = " + e);
    System.out.println("!a&b|a&!b = " + f);
    System.out.println("       !a = " + g);
  }
}
```

```
        a = true
        b = false
      a|b = true
      a&b = false
      a^b = true
a&b|a&!b = true
       !a = false
```

Q.2 a) Write a Java program to print following pattern.

```
0
1  2
3  4  5
6  7  8  9
```

```
public class Main
{
        public static void main(String[] args) {
                int i, j, n=4;
      int k=0;
       for (i = 1; i <= n; i++)
       {
          for (j = i; j >= 1; j--)
          {
             System.out.print(k+" ");
             k++;
          }
       System.out.println();
       }
          }
```

Q.2 b) Write a program to find factorial of a number 5.

```
import java.util.Scanner;
public class JavaExample {

   public static void main(String[] args) {

       //We will find the factorial of this number
      int number;
      System.out.println("Enter the number: ");
      Scanner scanner = new Scanner(System.in);
      number = scanner.nextInt();

      long fact = 1;
      int i = 1;
      while(i<=number)
      {
         fact = fact * i;
         i++;
      }
      System.out.println("Factorial of "+number+" is: "+fact);
   }
}
```

Output:

```
Enter the number:
6
Factorial of 6 is: 720
```

Q. 3 a) How Java arrays are initialized and declared ,explain with an example

We declare an array in Java as we do other variables, by providing a type and name:

int[] myArray;

To initialize or instantiate an array as we declare it, meaning we assign values as when we create the array, we can use the following shorthand syntax:

int[] myArray = {13, 14, 15};

To use the array, we can initialize it with the new keyword, followed by the data type of our array, and rectangular brackets containing its size:

```
int[] intArray = new int[10];
intArray[0] = 22;
```

The following code initializes an integer array with three elements - 13, 14, and 15:

int intArray[] = {13, 14, 15};

One of the most powerful techniques that you can use to initialize your array involves using a for loop to initialize it with some values.

Let's use a loop to initialize an integer array with values 0 to 9:

```
int[] intAray = new int[10];

for (int i = 0; i < intArray.length; i++) {
   int_array[i] = i;
}
```

Q. 3b)Write a Java program to check entered number is prime or not.

```
public class PrimeExample{
 public static void main(String args[]){
  int i,m=0,flag=0;
  int n=3;//it is the number to be checked
  m=n/2;
  if(n==0||n==1){
   System.out.println(n+" is not prime number");
  }else{
   for(i=2;i<=m;i++){
    if(n%i==0){
     System.out.println(n+" is not prime number");
     flag=1;
     break;
    }
   }
   if(flag==0)  { System.out.println(n+" is prime number"); }
  }//end of else
}
}
```

Output:

```
3 is prime number
```

Q. 4 a) Explain why java is a strongly typed language.

Java Is a Strongly Typed Language

It is important to state at the outset that Java is a strongly typed language. Indeed, part of Java's safety and robustness comes from this fact. Let's see what this means. First, every variable has a type, every expression has a type, and every type is strictly defined. Second, all assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility. There are no automatic coercions or conversions of conflicting types as in some languages. The Java compiler checks all expressions and parameters to ensure that the types are compatible. Any type mismatches are errors that must be corrected before the compiler will finish compiling the class.

Q. 4 b) Explain steps to execute simple java program and role of JVM in execution

```
// Java Program to Illustrate Compilation and Execution
// Stages

// Main class
Class hello {

        // Main driver method
        public static void main(String[] args)
        {

                // Print command
                System.out.print("Hello World");
        }
}
```

Let us understand the real compilation and execution process.

Step 1: Let us create a file writing simple printing code in a text file and saving it with ".java" extension.

Step 2: Open the terminal and go to the Desktop directory using the below command as follows.

```
cd /Users/mayanksolanki/hello.java
```

tep 3: Let us try to compile our program with the below command

```
javac hello.java
```

Step 4: Lastly run it with the below command as follows:

```
java hello
```

output:

Hello World

Q. 5 a) Explain three OOP principles.
i)Inheritance ii)Polymorphism iii)Encapsulation

Encapsulation

This post provides the theoretical explanation of Encapsulation with real-life examples. For detailed explanation on this topic with java programs refer encapsulation in java with example.

Encapsulation is:

- Binding the data with the code that manipulates it.
- It keeps the data and the code safe from external interference

Looking at the example of a power steering mechanism of a car. Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. It even controls the power delivered by the engine to the steering wheel. But to the external world there is only one interface is available and rest of the complexity is hidden. Moreover, the steering unit in itself is complete and independent. It does not affect the functioning of any other mechanism.

Inheritance

This post provides the theoretical explanation of inheritance with real-life examples. For detailed explanation on this topic with java programs refer inheritance with examples and types of inheritance in java.

- Inheritance is the mechanism by which an object acquires the some/all properties of another object.
- It supports the concept of hierarchical classification.

For example: Car is a four wheeler vehicle so assume that we have a class FourWheeler and a sub class of it named Car. Here Car acquires the properties of a class FourWheeler. Other classifications could be a jeep, tempo, van etc. FourWheeler defines a class of vehicles that have four wheels, and specific range of engine power, load carrying capacity etc. Car (termed as a sub-class) acquires these properties from FourWheeler, and has some specific properties, which are different from other classifications of FourWheeler, such as luxury, comfort, shape, size, usage etc.

Polymorphism

This post provides the theoretical explanation of polymorphism with real-life examples. For detailed explanation on this topic with java programs refer polymorphism in java and runtime & compile time polymorphism.

- Polymorphism means to process objects differently based on their data type.
- In other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)
- This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.

Q. 5 b) Differentiate between narrowing and widening with proper example

- **Widening**

Widening, also known as *upcasting*, is a *conversion* that implictly takes place in the following situations -

- Widening takes place when a *smaller primitive type value* is automatically accommodated in a *larger/wider primitive data type*.
- Widening takes place when a *smaller primitive type value* is automatically accommodated in a *larger/wider primitive data type*. Let us see an example.

```
//Java - Example of Widening a smaller primitive value to a larger primitive type

class A
{
public static void main(String... ar)
{
        byte b=10;

        short s= b;             //byte value is widened to short
        int i=b;    //byte value is widened to int
        long l=b; //byte value is widened to long
        float f=b;//byte value is widened to float
        double d=b;             //byte value is widened to double

        System.out.println("short value : "+ s);
        System.out.println("int value : "+ i);
        System.out.println("long value : "+ l);
        System.out.println("float value : "+ f);
        System.out.println("double value : "+ d);
}
}
```

Output-

```
short value : 10
int value : 10
long value : 10
float value : 10.0
double value : 10.0
```

- **Narrowing**

Narrowing, also known as *downcasting/casting*, is a *conversion* that is explictly performed in the following situations -

- Narrowing a *wider/bigger primitive type value* to a *smaller primitive type value*.

Narrowing a bigger primitive value to a small primitive value.

Narrowing is explictly performed when a *wider/bigger primitive type value* is assigned to a *smaller primitive type value*. This also known as **downcasting/casting** a bigger primitive value to a small primitive value. Let us see an example.

//Java - Example of narrowing a bigger primitive value to a small primitive value

```
class A
{
public static void main(String... ar)
{
        double d =10.5;

        byte b = (byte)d;           //Narrowing double to byte
        short s= (short)d;          //Narrowing double to short
        int i= (int)d;                      //Narrowing double to int
        long l= (long)d;            //Narrowing double to long
        float f= (float)d;          //Narrowing double to float


        System.out.println("Original double value : " +d);
        System.out.println("Narrowing double value to short : "+ s);
        System.out.println("Narrowing double value to int : "+ i);
        System.out.println("Narrowing double value to long : "+ l);
        System.out.println("Narrowing double value to float : "+ f);
        System.out.println("Narrowing double value to byte : "+ b);
}
}
```

Output-

```
Original double value : 10.5
Narrowing double value to short : 10
Narrowing double value to int : 10
Narrowing double value to long : 10
Narrowing double value to float : 10.5
Narrowing double value to byte : 10
```

Q. 6 a)) Explain following with syntax and example. i)for-each ii) for loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop.

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample {
public static void main(String[] args) {
   //Code of Java for loop
   for(int i=1;i<=3;i++){
       System.out.println(i);
   }
}
}
```

Output :

```
1
2
3
```

Java For-each Loop | Enhanced For Loop

```
for(data_type variable : array | collection){
//body of for-each loop
}
//An example of Java for-each loop
class ForEachExample1{
```

```
public static void main(String args[]){
 //declaring an array
 int arr[]={12,13,14,44};
 //traversing the array with for-each loop
 for(int i:arr){
   System.out.println(i);
 }
}
}
```

Output:

```
12
12
14
44
```

Q.6 b)Write a Java program to find the average of the array elements

```
public class Exercise4 {
public static void main(String[] args) {

    int[] numbers = new int[]{20, 30, 25, 35, -16, 60, -100};
    //calculate sum of all array elements
    int sum = 0;
    for(int i=0; i < numbers.length ; i++)
     sum = sum + numbers[i];
    //calculate average value
     double average = sum / numbers.length;
     System.out.println("Average value of the array elements is : " + average);
  }
}
```

Sample Output:

```
Average value of the array elements is : 7.0
```

Q. 7 a) Explain the following operators in Java

a)>>> b)<< c)? d) % e) &=

Unsigned right shift operator

The unsigned right shift operator '>>>' do not use the sign bit to fill the trailing positions. It always fills the trailing positions by 0s.

Thus a >>> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
And b >>> 1 = 0111 1111 1111 1111 1111 1111 1110 0010

Bitwise Left Shift Operator (<<)

Left shift operator shifts the bits of the number towards **left** a specified number of positions. The symbol for this operator is <<. When you write **x<<n**, the meaning is to shift the bits of **x** towards left **n specified** positions.

If **x=10**, then calculate **x<<2** value.

Which is 20

```
public class OperatorShifting

{

public static void main(String args[])

{

byte x, y;

x=10;

y=-10;

System.out.println("Bitwise Left Shift: x<<2 = "+(x<<2));

System.out.println("Bitwise Right Shift: x>>2 = "+(x>>2));

System.out.println("Bitwise Zero Fill Right Shift: x>>>2 = "+(x>>>2));

System.out.println("Bitwise Zero Fill Right Shift: y>>>2 = "+(y>>>2));

}
```

```
}
```

Output:

```
Bitwise Left Shift: x<<2 = 40
Bitwise Right Shift: x>>2 = 2
Bitwise Zero Fill Right Shift: x>>>2 = 2
Bitwise Zero Fill Right Shift: y>>>2 = 1073741821
```

Ternary Operator Java

In Java, the **ternary operator** is a type of Java conditional operator.

The meaning of **ternary** is composed of three parts. The **ternary operator (? :)** consists of three operands. It is used to evaluate Boolean expressions. The operator decides which value will be assigned to the variable. It is the only conditional operator that accepts three operand

```
public class TernaryOperatorExample

{

public static void main(String args[])

{

int x, y;

x = 20;

y = (x == 1) ? 61: 90;

System.out.println("Value of y is: " +  y);

y = (x == 20) ? 61: 90;

System.out.println("Value of y is: " + y);

}

}
```

Output

Value of y is: 90
Value of y is: 61

Modulo or Remainder Operator returns the remainder of the two numbers after division. If you are provided with two numbers, say A and B, A is the dividend and B is the divisor, A mod B is there a remainder of the division of A and B. Modulo operator is an arithmetical operator which is denoted by %.

Input : a = 15, b = 6
// 15%6 means when we divide 15(numerator) by 6(denominator) we get remainder 3//
Output: 3

&= Bitwise AND assignment
Int a=3;
Int b=6;

a&=b; // a= a&b
System.out.println(a); // 2