Introduction to JAVA

Sudhamshu B N
1DS18EC091

1.)

A) b) James Gosling

B) a) Bytecode executed in JVM

C) c) Use of pointers

D) a) if

E) b) char [] c = new char[5];

F) a) true / false boolean data

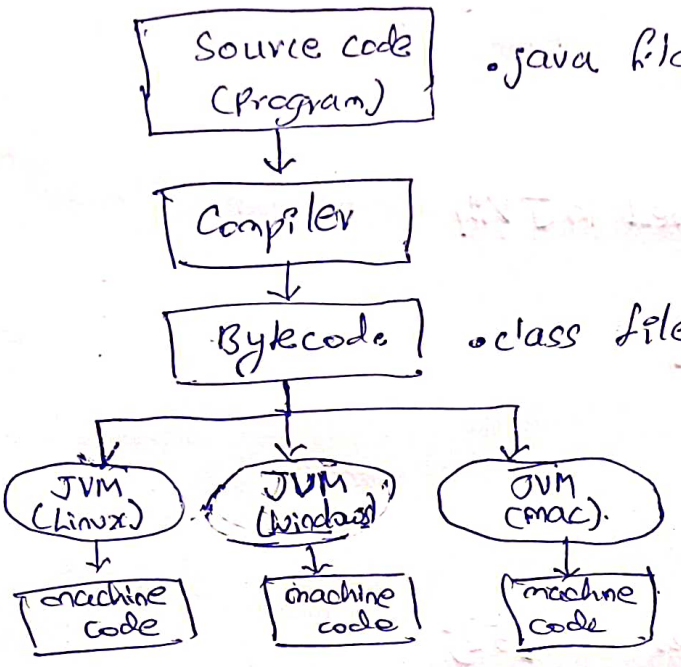
G) a) class

H) b) memory used by the object with no reference is automatically reclaimed.

I) a) final

J) b) public

2-) A) Byte code



- Byte code (.class file) is highly optimized set of instructions.
- They are designed to run in Java run time system. (JVM) which is Java virtual machine.
- Translating Java program into bytecode makes it platform independent.
- It also helps to keep it secure.

Advantages
- platform independent
- portability unlike C/C++

B) i) Simple
ii) Secure
iii) Portable
iv) Object Oriented.
v) Robust
vi) Multithreaded.
vii) Architecture Neutral.

i) Simple: Java is simple to code. like C/C++ because it inherits C/C++ syntax and many object oriented features like C++.

ii) Secure: Bytecode is executed in JVMs (Java virtual machines) which makes it fully secured under control of JVM.

Portable: Java can be run in any system which has JVM, unlike C/C++, which means it has ability to run program without depending on underlying hardware/ O.S.

Object oriented: The model used in JAVA is reusable which is called object oriented. which can be extented easily to further programs/functionalities.

Robust: The below properties make it robust,
- Platform independent
- OOP Language
- memory management
- Exception handling.

Multithreaded: Allows to develop program that can do multiple tasks simultaneously. Not unique to JAVA, but important property.

Architecture-Neutral: Java can be run in any OS which has Java virtual machine. It can have any computer architecture which makes it architecture neutral.
Ex:- Java can be run in windows, Linux & mac).

3.) Constructors: Every class has a constructor (default constructor class) Constructor should have same name as class.

- No argument constructor
- Parameterized constructor

*No argument constructor:
Constructor which has no argument

for example.
→ public constructor() {
// body
}

- In the below example, we have constructor main(), which has no parameters.

constructor is declared as private here, which can also be declared as public which makes main public.

example 26

```
class main{
    int i;
    // constructor with no parameter
    private main(){
        i=5;
        System.out.println("constructor is called");
    }
    public static void main(String[] args){
        // calling constructor without any parameter
        main obj = new main();
        System.out.println("value" + obj.i);
    }
}
// Output
// constructor is called
// value 5
```

- In the below example

* parameterized constructor

is explained.

Here we have main() which takes single parameter.

notice, main obj1 = new main("Java");

Here the language variable is initialized.

exe:-

```
class main {
    String languages;
    // constructor with single value
    main (String lang) {
        languages = lang;
        System.out.println (languages + " language");
    }
    public static void main (String[] args) {
        // call constructor by passing single value
        main obj1 = new main ("Java");
        main obj2 = new main ("Python");
    }
}
// output
// Java language
// Python language
```

4.) A) Access Specifiers

- private
- protected

- **Private:**
  - → Access level of private modifier is within the class
  - → It can't be accessed outside the class
  - → It can't be accessed within package / outside package by subclass / outside package etc.
  - → only accessed within the class

ex:-

```
class privateexample {
    private int data = 40;
    private void message() {
        System.out.println("Hello, Sudhanshu here");
    }
public class Sample {          // can't be accessed.
    public static void main (String args[]) {
        privateexample obj = new privateexample();
        System.out.println(obj.data); // Error, compile time
        obj.message(); // compile time error
    }
}
```

- **Protected**: Access level is within the package & outside the package through sub class

ex:-

```
package pack;
public class protectedexample {
    protected void message() {
        System.out.println("Hello, Sudhanshu here");
    }
}
package newpath; // using subclass
import pack;
class protectedexample1 extends protectedexample {
    public static void main (String args[]) {
        protectedexample1 obj = new protectedexample2();
        obj.message();
    }
}
```

o/p: Hello, Sudhanshu here.

4) b) Static keyword is used for memory management mainly. It can be applied with variables & nested classes

Static can be
- → Variable
- → Method
- → Block
- → Nested class

- It can be used to refer to common property of all objects for example company name of employees, college name etc
- It gets memory only once.
- It is memory efficient

ex:
```
class StaTest{
    int rollnos; // instance var        // Static college string
    String names;
    static String college = "DSCE"; //Static
    //constructor
     Student (int r, String n){
      rollno = rs
       name = ns
      }
//method to display values
    void display(){
      System.out.println( rollno + " " + name + " " + college);}
}
public class Test{
    public static void main (String args[]){
     Student S1 = new Student(091, "Sudharshu");
     S1.display();
    }}
```
Static

O/P: 091 Sudharshu DSCE

6.) A)

i) Unsigned right shift operator

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

Exc:-

```
public class test{
    public static void main (String[] arg)
    int a = 60  int b = 0;
    System.out.println ("60 = " + Integer.toBinaryString(a));
    // unsigned shift
    b = a >>> 1;
    System.out.println ("60 >>> 1 = " + Integer.toBinaryString(b));
    }
}
```

//output

// 60 = 111100

// 60 >>> 1 = 11110

ii) Break statement

when break encountered loop immediately terminated and control resumes to next statment following the loop.

exc:-

```
public class test{
    public static void main (String args[]){
        int[] numbers = {1, 2, 3, 4, 5};
        for (int x: numbers){ if (x==3){ break;}}
        System.out.println(x);
    }}
```

o/p: 1

2

B.) Method overloading

In JAVA two or more methods may have same name if they differ in parameters: (Number & Type).

These methods are overloaded

for example:

```
void func() { // statements }
void func(int a) { // statements }
```

here the func() method is overloaded. This may happen due to number of arguments also by parameters

for example

```
class methodoverloading {
    private static void display(int a){
        System.out.println("Argument "+a); }
    private static void display(int a, int b){
        System.out.println("Arguments"+a+"and"+b);}
    public static void main(String[] args){
        display(1);
        display(1,4);
    }
}
```

O/P

// Argument 1 ← overloaded.

// Arguments 1 and 4