Name : Sabika Shabbir Naikawadi

1. What do you know about JVM, JRE, & JDK?

JDK - Java Development Kit.

- JDK provides ~~an~~ environment ~~for~~ tools for developing, compiling, debugging & executing Java programs.
- JDK is a platform dependant software. ~~that we can~~
- In order to develop Java application we need JDK in our system.
- JDK = Development tools + JVM + JRE

JRE - Java Runtime Environment

- JRE is a platform dependant software.
- JRE is a bundle of software components used to run Java applications.
- In order to run a Java application we need ~~to~~ JRE.
- It consists of JVM, and the classes required to run a Java program.

JVM - Java Virtual Machine

- JVM is responsible for execution of Java program.
- JVM first interprets the byte code, then store the class information in the memory. Then it executes the bytecode generated by java compiler.

2.) Is JRE platform dependant or independant?

— JRE is platform dependant as it requires different configuration for different OS.

3) Which is the ultimate Base class in java class hierarchy? List the name of methods of it?

→ - The ultimate base class in java class heirarchy is Object class.

- Object class is a concrete class in java.lang package.
- java.lang.Object do not extend any and do not implement any interface.
- It only contains parameterless constructors

- There are 11 methods in Object class

1. toString();
2. equals(Object obj);
3. hashCode();
4. clone();
5. finalize();
6. getClass();
7. wait()
8. wait(long timeout);
9. notify();
10. notifyAll();
11. wait(long timeout, int nanos);

4 Which are the reference types in Java?

— - In Java, non primitive types are called as Reference types
- As variables of the non primitive data types contains reference of the instance.
- There are four reference types
  1. Class
  2. Array
  3. Enum
  4. Interface

5. Explain Narrowing & Widening?

— ◦ Widening (Upcasting)

- When we need to convert the value of variable of (smaller) narrower ~~data~~ type into the (larger) wider type, then this process is called Widening

- Example:

```
byte a = 4;
int b = a;
```

can also be written as

```
byte a = 4;
int b = (int) a;
```

- In widening conversion, we don't need to explicitly typecast.

◦ Narrowing (Downcasting)

- The process of converting the value of variable of (larger) wider type to (smaller) narrower type is called Narrowing conversion.

- In case of narrowing, explicit type casting is mandatory.

Example : double num1 = 10.5;
int num2 = (int) num1;

- Here, the value of num1 will be cut down to 10 and stored in num2.

6. How will you print "Hello CDAC" statement on screen, without semicolon?

- class Solutions
{
    public static void main (String[] args)
    {
        if (System.out.printf ("Hello CDAC")==null)
        {
        }
    }
}

7. Can you write java application without main function? If yes, how?

→ - ~~No~~ No since JDK 1.7 it is not possible to execute any java class without main() method.

- But till JDK 1.6 it ~~was~~ possible to write a java program without main method.

Example :

```
class MainDemo {

    static
    {
        System.out.println ("Inside Static Block.");
        System.exit(0);
    }
}
```

output ~~for~~ (JDK 1.6): Inside Static Block.

- But from JDK 1.7 the above code gives an error in output
- ERROR: main method not found in class MainDemo

8 What will happen, if we call main method in static block?

— - The static block gets executed when the class gets loaded, & the class gets loaded because it contains the main method.
- We have no idea when the ~~#~~ static blocks are executed exactly.
- ~~It will~~
- When we call main method in static block it will work but it ~~is~~ ~~& not~~ should not be done.
- Static blocks ~~are~~ should only be used to initialize variables. ~~or~~

```
class Demo{

  static{
        System.out.println("\n Static Block");
        main(null);
      }
public static void main(String[] args)
{
    System.out.println("\n Main");
}
}
```

o/p: \n Static Block
\n Main
\n Main → main method executed

Static block executed

9. In System.out.println, Explain meaning of each word?

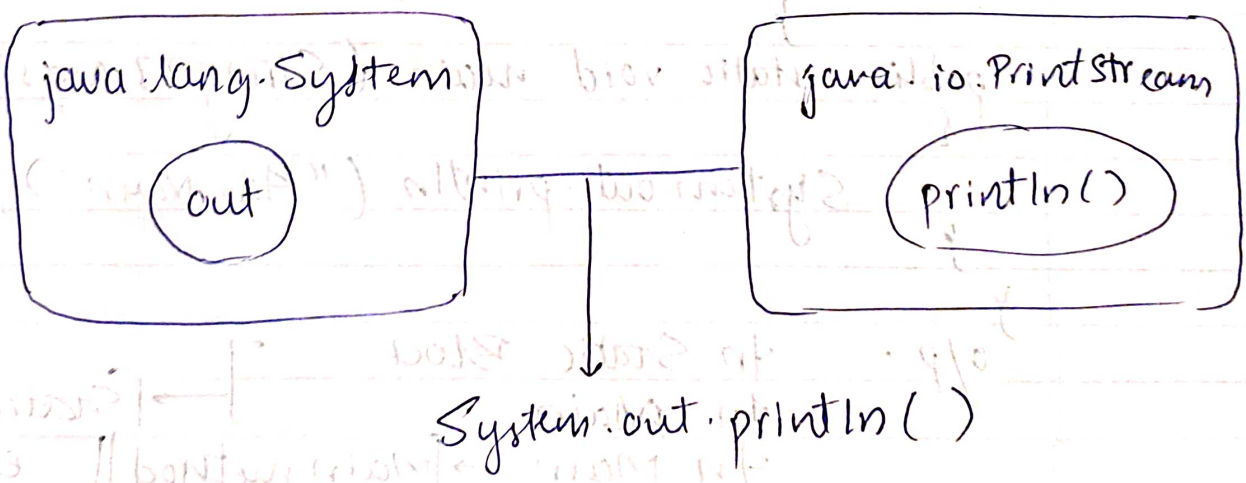
→ Java System.out.println() is used to print argument passed to it.

(1) System : It is a final class defined in the java.lang package.

(2) ~~out : This is an instance of PrintStream type, which is a public & static member field of the System class.~~

(2) out : out is reference of java.io.PrintStream class. It is declared as static final field inside System class

(3) println : println is a non static method of java.io.PrintStream class.



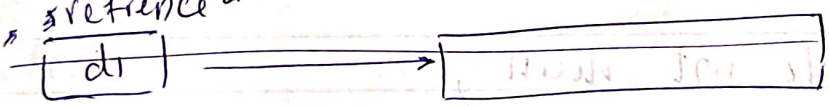
10. How will you pass object to the function by reference?

→ - In Java all objects are dynamically stored in Heap Space.
- These ~~of~~ objects are referred through reference variables.
- Whenever ~~we~~ the object is passed as an argument, an exact copy of the reference variable is created which points the original object.
- As a result, whenever we any changes in the same object in the method, that change is reflected in the original object.
- Example.

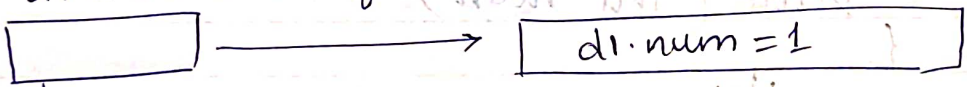
```
class Demo {

    public int num;

    Demo (int num)
    {
        this.num = num;
    }
}
public class Main
{
```

```
public static void main (String [] args)
{
    Demo d1 = new Demo(1);
    Demo d2 = new Demo (1);

    modify ( d1 , d2);

}

public static void modify (Demo a , Demo b)
{
    a = a.num++;

    b = new Demo (1);
    b.num ++;
}

} // main class closed
```

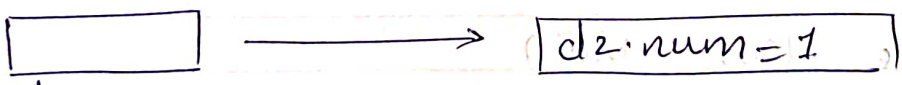
- Here, reference



- Here, d1 & d2 are reference variables



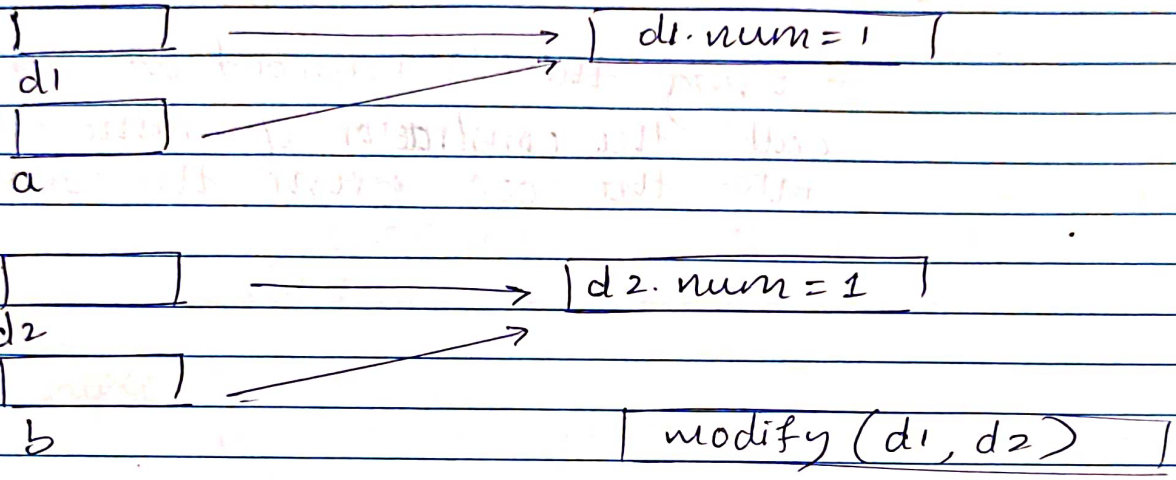
Demo d1 = new Demo (1);



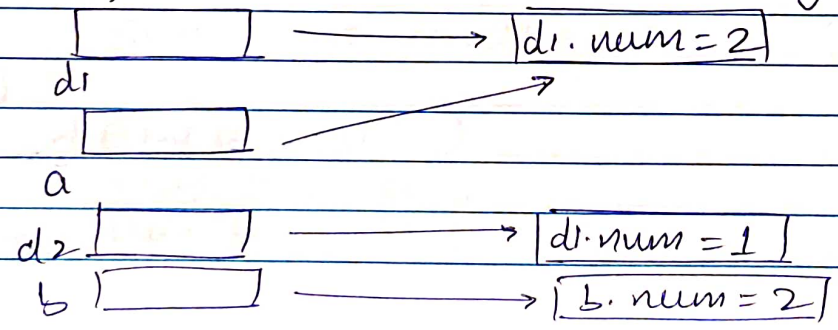
Demo d2 = new Demo (1);

- Here both d1 & d2 are pointing to two distinct objects locations.

- When we pass d1 & d2 in the method modify() method, it creates mirror copies of those references a & b which point to the same old objects



- In modify() method when we modify reference a it changes the original object.
- But for reference b we have assigned a new object, so its now pointing to new object.



11 Explain Constructor Chaining? How can we achieve it in C++?

→ Constructor Chaining

- The process of calling one constructor from another constructor is called as Constructor Chaining.
- This can be done in two ways.

1. this(): By using this() ~~the~~ for chaining constructors in the same class.

2. super(): By using super() for chaining constructors from the parent class.

Example of this():

```
class A{   int a, int b;
    A()
    {   this(10, 20);
        System.out.println(a + "  " + b);
    }
    A(int a, int b)
    {   this.a = a;
        this.b = b;
}   }
```

→ Here ~~zero~~ A() calls the two_arg constructor A(int a, int b).

Example of super():

```
class First
{
    String str;

    First()
    {
        this("");
        System.out.println("zero arg constructor");
    }
    First(String s)
    {
        this.str = s;
    }
}
class Second extends First
{
    Second(String str)
    {
        super(str);
        System.out.println("calling super constructor");
    }
}
```

We can achieve constructor chaining in C++ using constructor initializer list

Example

```
#include <iostream>
using namespace std;

class A
{   int x, y, z;
    A()
    {   x = 0;
        y = 0;
        z = 0;
    }

    A(int z) : A()
    {   ~~cout << x << y << z;~~
        this->z = z;  cout << x << y << z;
    }
};
int main()
{   A obj(3);
    obj return 0;
}
```

1.] within same class constructor initializer list

- from C++ 11 ~~has~~ we get this feature ~~C++11~~ called constructor delegation.

12 Which are the rules to overload method in sub class?

→ Rules to overload methods in sub class are similar to when we overload within a same class.

Rules.

1.) Methods must have same method name.
2.) Methods must have different gargument lists.
3. The methods must follow above two rules to overload, they may or may not
3.) Have different return types
4.) Have different access modifiers.
5.

Example :

```
class A
{
    void show ()
    {
        System.out.println ("Hello World");
    }
}
class B extends A
{
    void show (int a)
    {
        System.out.println (a);
    }
}
```

13. Explain the difference among finalize & dispose?

→ finalize()

- finalize() method is defined in an Object class
- It is called by garbage collector on the objects that aren't referenced anymore & have been selected for garbage collection
- The syntax of finalize():

```
protected void finalize()
{

}
```

- It is invoked by garbage collector.

dispose()

- dispose() method is defined in IDisposable interface.
- It is used to release unmanaged resources stored by an object, like files or streams.
- Syntax:

```
public void dispose()
{

}
```

- dispose() method is invoked by user.
- It is invoked faster than finalize()

14 Explain the difference among final, finally & finalize?

→ 1. final

- final is a keyword in Java.
- we can use final keyword with
  - class
  - method
  - variable
  - method parameter declarations
- Making a class final means that it won't be possible to extend that class
- Making a method final, we cannot override that method.
- Making a variable or parameter means that once the reference has been assigned then it cannot be changed.

Examples

① class

```
final class Test
{
    int a;
    void myFun()
    {
        System.out.println("a");
    }
}
```

(2) final with method & variable

```
final void add(int a, int b)
{
    final int c = 10;
    a=5;
}
```

Here add(int a, int b) is a final method which cannot be overriden. The value of c which is a final variable cannot be (changed)/modified.

2. finally

- finally block is an optional block to use with a try/catch statement.
- In this block, we write the code to be executed after the try/catch structure, whether an exception is thrown or not.
- We can also use finally block without a catch block after try block.
- The code will then be executed after try block.

Example:

```
public static void main(String[] args)
{
    try{
        System.out.println("Try Block");
        throw new Exception();
    } catch (Exception e){
        System.out.println("Execute catch");
    }
finally
{
    System.out.println("finally block");
}
```

3. Finalize

- finalize method is a protected method defined in an Object class.
- It's called by the garbage collector on objects that aren't referenced anymore & have been selected for garbage collection.
- We can override this method to define the behaviour an object must have when collected by the garbage collector.

15. Explain the difference among checked & unchecked exception?

→ (1.) Checked Exceptions.

- Checked exceptions represents errors outside the control of the program.
- Java verifies the checked exception at compile time.
- Therefore, we should use throws keyword to declare a checked exception.
- Some common checked exceptions in Java are IO Exception, SQL Exception & ParseException.
- We can also use try-catch block to handle checked exception.
- The Exception class is the super class of checked exceptions, so we can create custom checked exceptions by extending Exception

(2.) Unchecked Exceptions

- Unchecked Exceptions represent the logical errors.
- Example.
  If we divide a number by zero, Java will throw ArithmeticException.
- Java does not check the unchecked exceptions at compile time.
- We don't have to declare unchecked exceptions in methods with the throws keyword.

- Some common unchecked exceptions in Java are NullPointerException, ArrayIndexOutOfBound Exception & IllegalArgumentException.
- RuntimeException class is the super class of all unchecked exceptions.
- We can create custom unchecked exception by the RuntimeException.

16. Explain Exception chaining?

→ - Chained exception helps to identify a situation in which one exception causes another Exception in an application.
- For example consider a method which throws ArithmeticException because of an attempt to divide by zero but the actual cause of exception was an I/O error which caused the divisor to be 0.
- the method will throw the ArithmeticException to the caller. The caller would not know about the actual cause of Exception.
- Exception Chaining is used in such situations.
- This concept was introduced in JDK 1.4.
- Throwable class has some constructors and methods to support exception chaining.

Constructors

1. Throwable (Throwable cause)
2. Throwable (Throwable cause
3. Throwable (String desc, Throwable cause)

Methods

1. getcause()
2. initcause()

17. Explain the difference between throw & throws?

→ 1.) throw

- The throw keyword in Java is used to explicitly throw an exception from a method & any block of code.
- The throw keyword is mainly used to throw custom exceptions.
- The throw keyword is used to throw an exception explicitly.
- It can throw only one exception at a time.

Example:

```
class ThrowExp {
    static void fun()
    {
        try {
            throw new NullPointerException("demo");
        } catch (NullPointerException e)
        {
            System.out.println("Caught inside fun()");
            throw e;
        }
    }
public static void main (String[] args)
{
    try
    {
        fun(); }
```

```
catch (NullPointerException e)
{   System.out.println("Caught in main");
}
}
}
```

2) throws

- throws is a keyword in Java used in signature of the method to indicate that this might throw one of the listed type exceptions.
- The caller to these methods has to handle the exception using a try-catch block
- throws may throw multiple exceptions seperated by comma.
- Whichever exception occurs if matched with declared ones then is thrown automatically.
- Used in case of checked Exceptions only.

Example:

```
public static void execute() throws
                    SocketException, ConnectionException,
                    Exception
```

18. In which case, finally block doesn't execute?

→ - If JVM exits while the try or catch code is being executed, then finally block may not execute.

- When the System.exit() method is called in the try block before the execution of finally block, finally block will not be executed.

```
class FinallyBlock
{
    void fun()
    {
        try
        {
            System.out.println("try");
            System.exit(0);
        }
        finally
        {
            System.out.println("finally");
        }
    }
    public static void main(String[] args)
    {
        FinallyBlock f = new FinallyBlock();
        f.fun();
    }
}
```

19. Explain up casting?

→

UP CASTING

- The process of converting the reference of sub class to reference of super class is called as upcasting.
- If we ~~to~~ want to minimize the instance dependancy then we should use up casting.
- Super ~~cla~~ class reference variable can contain the reference of sub class object.

Example.

```
class A
{
    void fun()
    {
        System.out.println("Hello");
    }
}
class B extends A
{
    void fun1()
    {
        System.out.println("World");
    }
}
public class Main {
  public static void main(String[] args)
{
        A obj = new B();
        obj.fun();
```

20. Explain dynamic method dispatch?

→ Dynamic method dispatch.

- The process of calling the s method of subclass using the reference of variable of super class is called dynamic method dispatch.

-Example.

```
class Person{
    public void print()
    {
        System.out.println("Person");
    }
}
class Employee extends Person{
    public void print()
    {
        System.out.println("Employee");
    }
}
class Main{
    public static void main(String[] args)
    {
        Person p = new Employee();
        p.print();
    }
}
```

dynamic method dispatch

upcasting

21. What do you know about final method?

→ Final method

- final methods cannot be overriden
- we cannot redefine final method in subclass.
- final methods get inherited inside the sub class.
- ~~Syntax~~ Example

```
final void fun_name ( ) {
}
```

- we can make an overriden method final.

22. Explain fragile base class problem and how can we overcome?

→

- Fragile Base class is a common problem with inheritance, which applies to Java and any other language that supports inheritance.
- The base class is considered fragile because changes to this class can have unexpected results in the classes that inherit from it.
- There are some methods to prevent this, but no straightforward method entirely avoid it while still using inheritance.

- You can prevent other classes inheriting from a class by labelling the class declaration final in Java.

23. Why Java does not support multiple implementation inheritance?

- In C++, combination of Heirarchical & multiple inheritance arise some problems called Diamond problem.
- To avoid this diamond problem, Java do not support multiple implementation inheritance
- In Java, a class can extend only one class.

24 Explain Marker Interface? List the name of some marker interfaces?

→ Marker Interface - It is an empty interface (no fields & methods)

Some of the marker interfaces are

- Serializable interface
- Cloneable interface
- Remote interface

```
public interface Serializable{

}
```

- Cloneable interface is present in java.lang package.
- Serializable interface is present in java.io package.
- Remote interface is present in java.rmi interface

25 Explain Significance of Marker Interface?

→ - A marker interface is an interface that doesn't have any methods or constants inside it. It provides run-time type information about the objects, so the compiler and JVM have

additional information about the object.

- A marker interface is also called a tagging interface.