Horipriya Das Arpita - 2019831068

এক্সামে কমন পড়িলে ট্রিট দিয়েন 😐

Java Question Answer

1. Write the output of the following java program-

Public static void main (String args[]){

int i = 0;

i = i++ + i;

System.out.println("i= "+i);

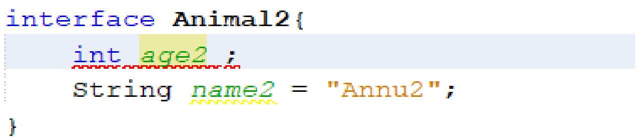
}

Output: i = 1

2. Does java support multiple inheritance? If yes, how and if not, why?

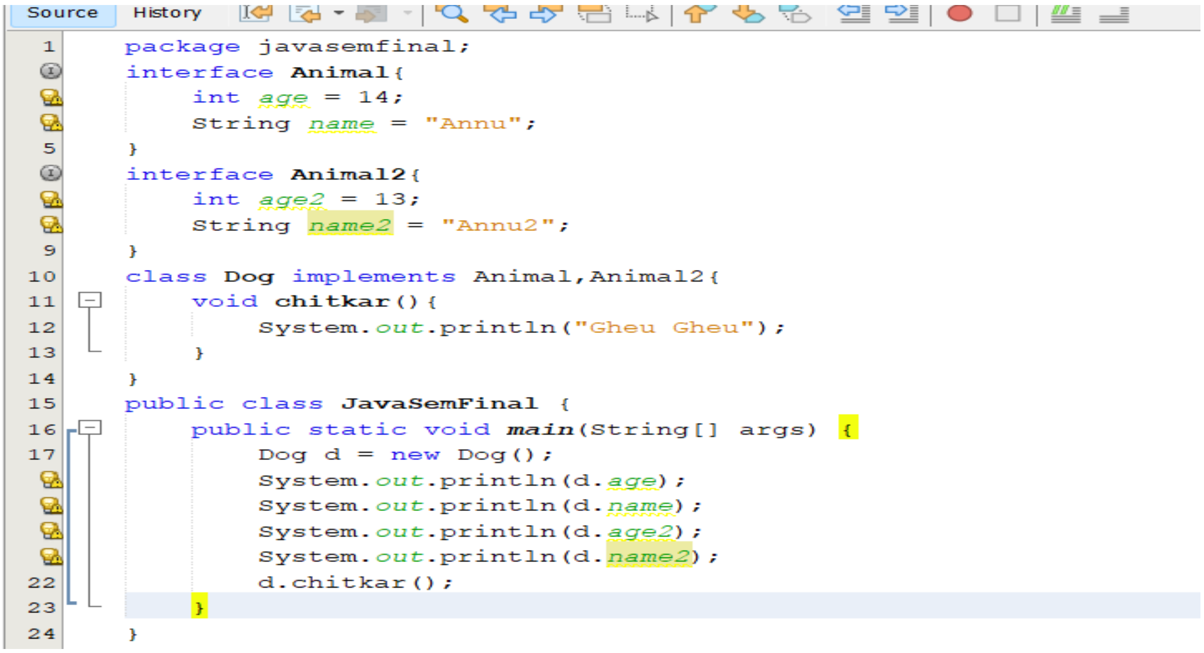
Ans: No, JAVA doesn't support multiple inheritance. This means you can't extend two or more classes in a single class. The reason behind this is to prevent ambiguity. Consider a case where class C extends class A and class B. Both class A and class B have a common method named display(). Now, Java compiler cannot decide which display method it should inherit. To prevent such situation java doesn't allow multiple inheritance.

When we need to extend two or more classes in Java, we need to refactor the classes as interfaces.



```
interface Animal2{
    int age2 ;
    String name2 = "Annu2";
}
```

This is not allowed. We have to initialize first. This is because Java allows implementing multiple interfaces on a single class.



```
package javasemfinal;
interface Animal{
    int age = 14;
    String name = "Annu";
}
interface Animal2{
    int age2 = 13;
    String name2 = "Annu2";
}
class Dog implements Animal,Animal2{
    void chitkar(){
        System.out.println("Gheu Gheu");
    }
}
public class JavaSemFinal {
    public static void main(String[] args) {
        Dog d = new Dog();
        System.out.println(d.age);
        System.out.println(d.name);
        System.out.println(d.age2);
        System.out.println(d.name2);
        d.chitkar();
    }
}
```

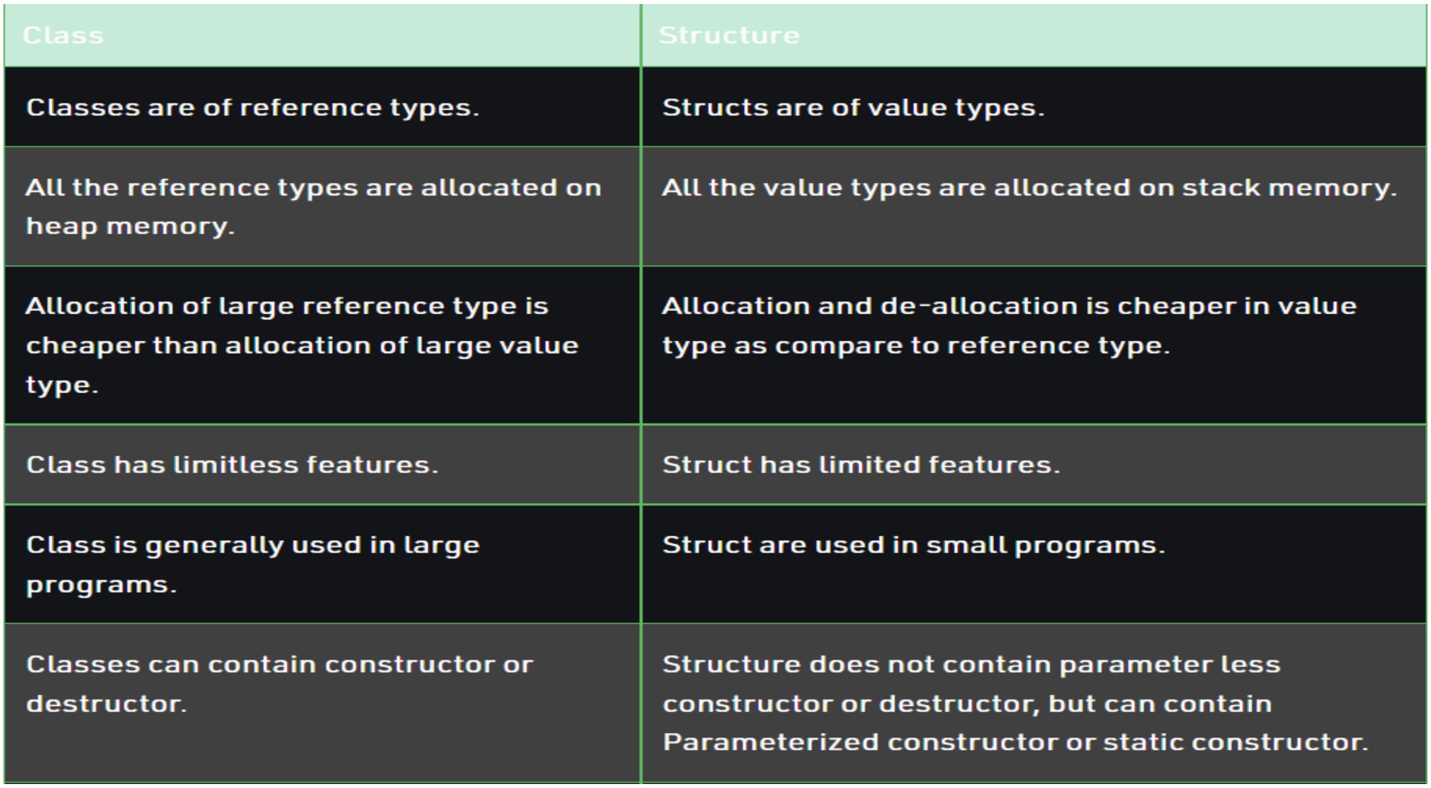
3. Can constructor be inherited?

Ans: Constructor is a block of code that allows you to create an object of class and has same name as class with no explicit return type.

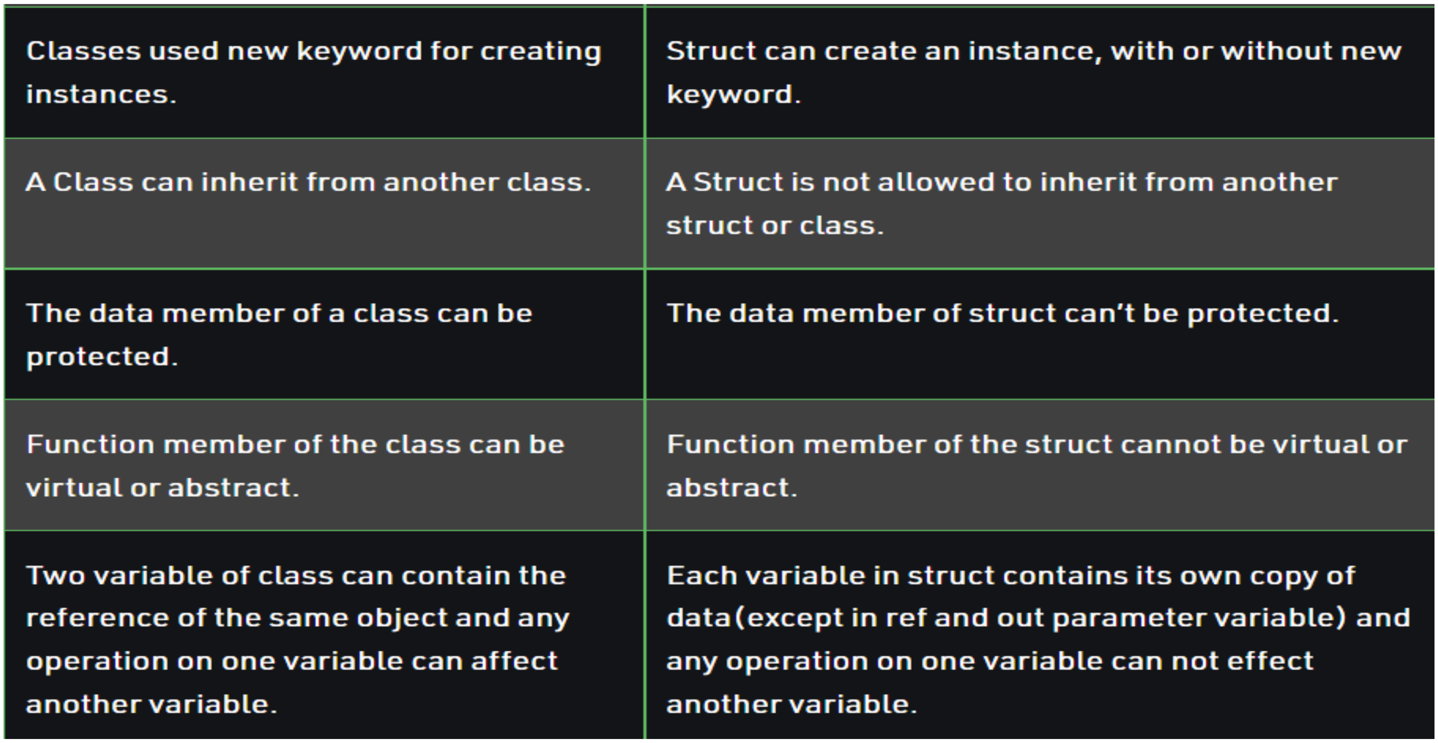
Whenever a class (child class) extends another class (parent class), the sub class inherits state and behavior in the form of variables and methods from its super class but it does not inherit constructor of super class because of following reasons:

- Constructors are special and have same name as class name. So, if constructors were inherited in child class, then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name. If we define Parent class constructor inside Child class it will give compile time error for return type and consider it a method. But for print method it does not give any compile time error and consider it an overriding method.
- Now suppose if constructors can be inherited then it will be impossible to achieving encapsulation. Because by using a super class's constructor we can access/initialize private members of a class.
- A constructor cannot be called as a method. It is called when object of the class is created so it does not make sense of creating child class object using parent class constructor notation. i.e. Child c = new Parent();
- A parent class constructor is not inherited in child class and this is why super() is added automatically in child class constructor if there is no explicit call to super or this.

4. Compare structure from C and class in java.



Class	Structure
Classes are of reference types.	Structs are of value types.
All the reference types are allocated on heap memory.	All the value types are allocated on stack memory.
Allocation of large reference type is cheaper than allocation of large value type.	Allocation and de-allocation is cheaper in value type as compare to reference type.
Class has limitless features.	Struct has limited features.
Class is generally used in large programs.	Struct are used in small programs.
Classes can contain constructor or destructor.	Structure does not contain parameter less constructor or destructor, but can contain Parameterized constructor or static constructor.



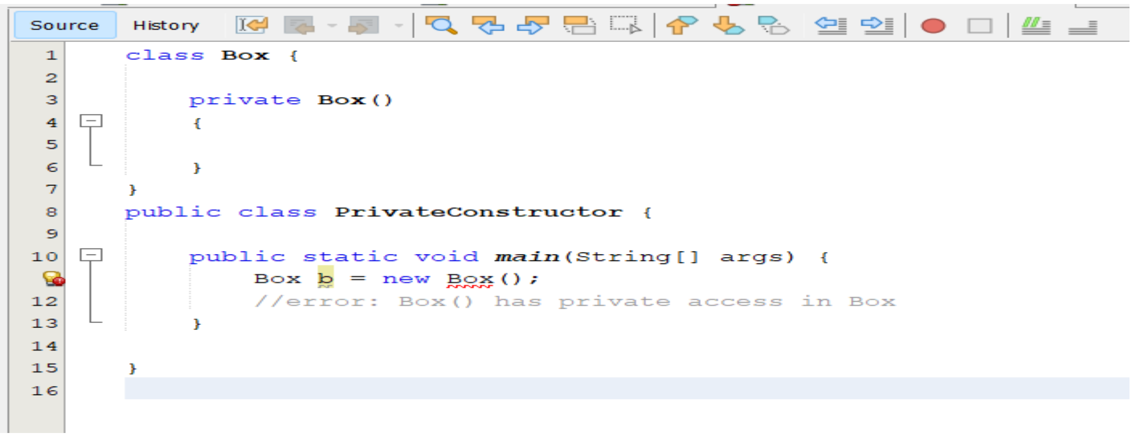
Classes used new keyword for creating instances.	Struct can create an instance, with or without new keyword.
A Class can inherit from another class.	A Struct is not allowed to inherit from another struct or class.
The data member of a class can be protected.	The data member of struct can't be protected.
Function member of the class can be virtual or abstract.	Function member of the struct cannot be virtual or abstract.
Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable.	Each variable in struct contains its own copy of data(except in ref and out parameter variable) and any operation on one variable can not effect another variable.

5. Can you declare a constructor private?

Ans: Yes, we can declare a constructor as private. But if we declare a constructor as private, we are not able to create an object of a class.

- A **private constructor** does not allow a class to be subclassed.
- A **private constructor** does not allow to create an object outside the class.
- If all the constant methods are there in our class we can use a **private constructor.**
- If all the methods are **static** then we can use a **private constructor.**
- If we try to **extend a class** which is having private constructor **compile time error will occur**.

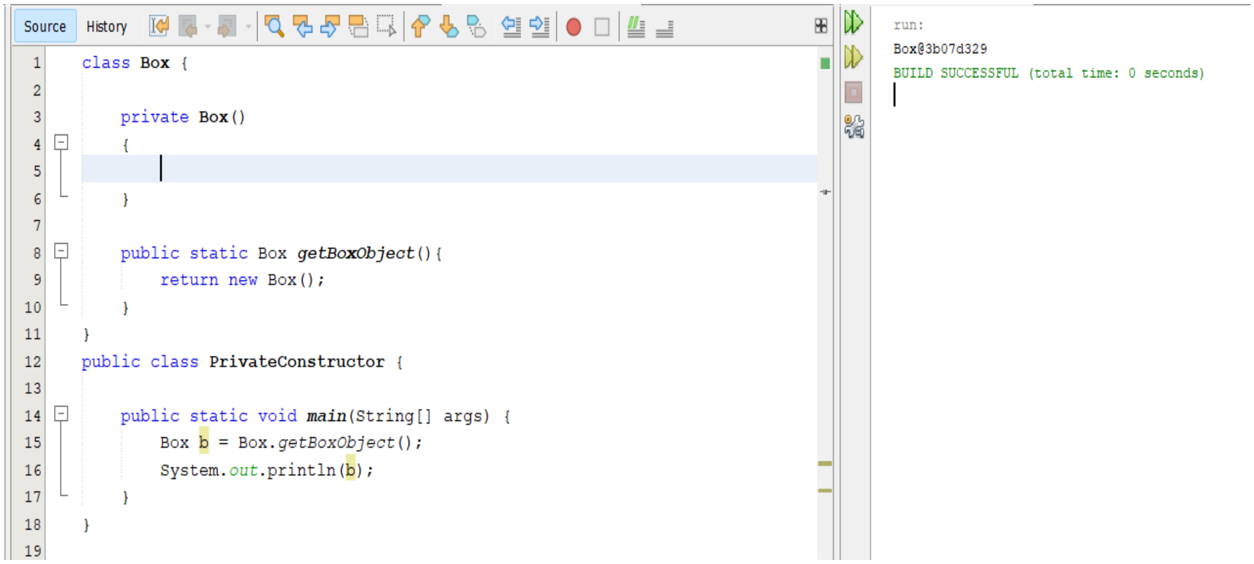
For example, here we have created a class named "Box" and made its constructor as private. Now if we want to make an object of this Box class in the main function, compile time error will occur.



```
class Box {

    private Box()
    {

    }
}
public class PrivateConstructor {

    public static void main(String[] args) {
        Box b = new Box();
        //error: Box() has private access in Box
    }

}
```

Here is the solution:

We can create a static method named getBoxObject() which return new Box. We can call this method to create an object of Box class. This will not show any error.

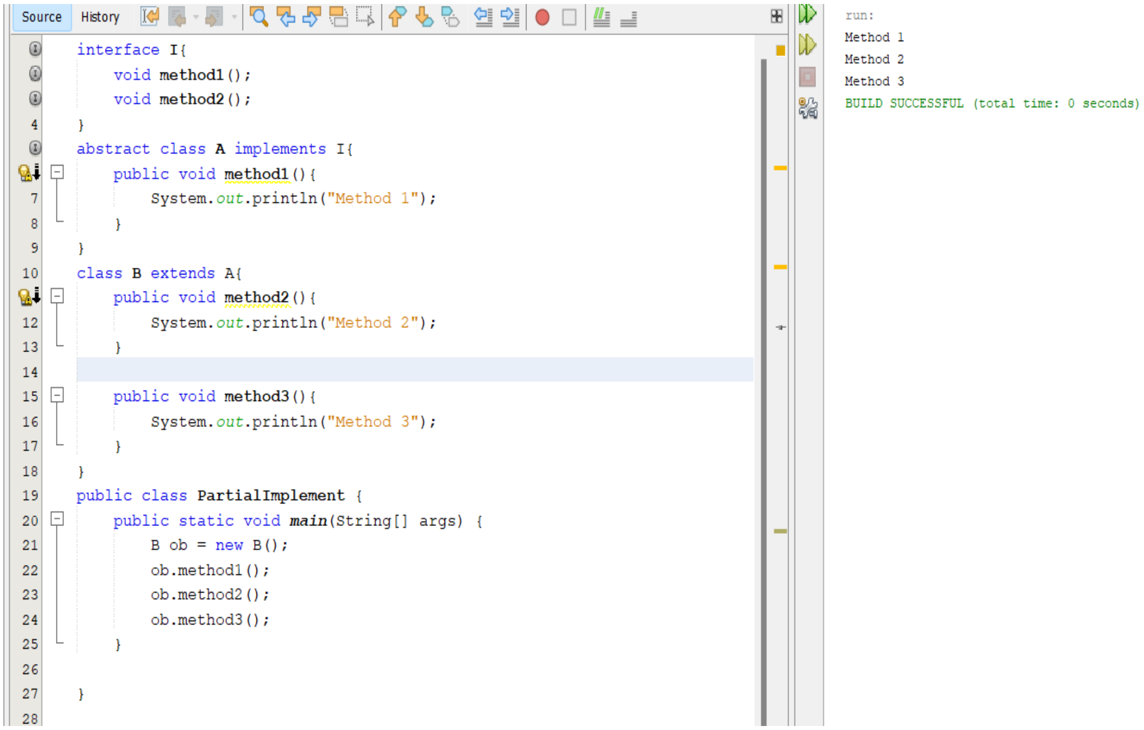


```
class Box {

    private Box()
    {
        
    }

    public static Box getBoxObject(){
        return new Box();
    }
}
public class PrivateConstructor {

    public static void main(String[] args) {
        Box b = Box.getBoxObject();
        System.out.println(b);
    }
}

```

```
run:
Box@3b07d329
BUILD SUCCESSFUL (total time: 0 seconds)
```

6. If a class want to partially implement an interface, what is the solution?

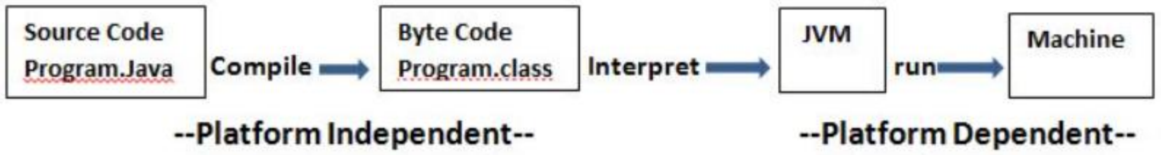
Ans: If a class want to partially implement an interface, we can create an abstract class that implements the interface. Any sub class of this abstract class can partially implement the interface.



```
interface I{
    void method1();
    void method2();
}
abstract class A implements I{
    public void method1(){
        System.out.println("Method 1");
    }
}
class B extends A{
    public void method2(){
        System.out.println("Method 2");
    }

    public void method3(){
        System.out.println("Method 3");
    }
}
public class PartialImplement {
    public static void main(String[] args) {
        B ob = new B();
        ob.method1();
        ob.method2();
        ob.method3();
    }

}

```

```
run:
Method 1
Method 2
Method 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

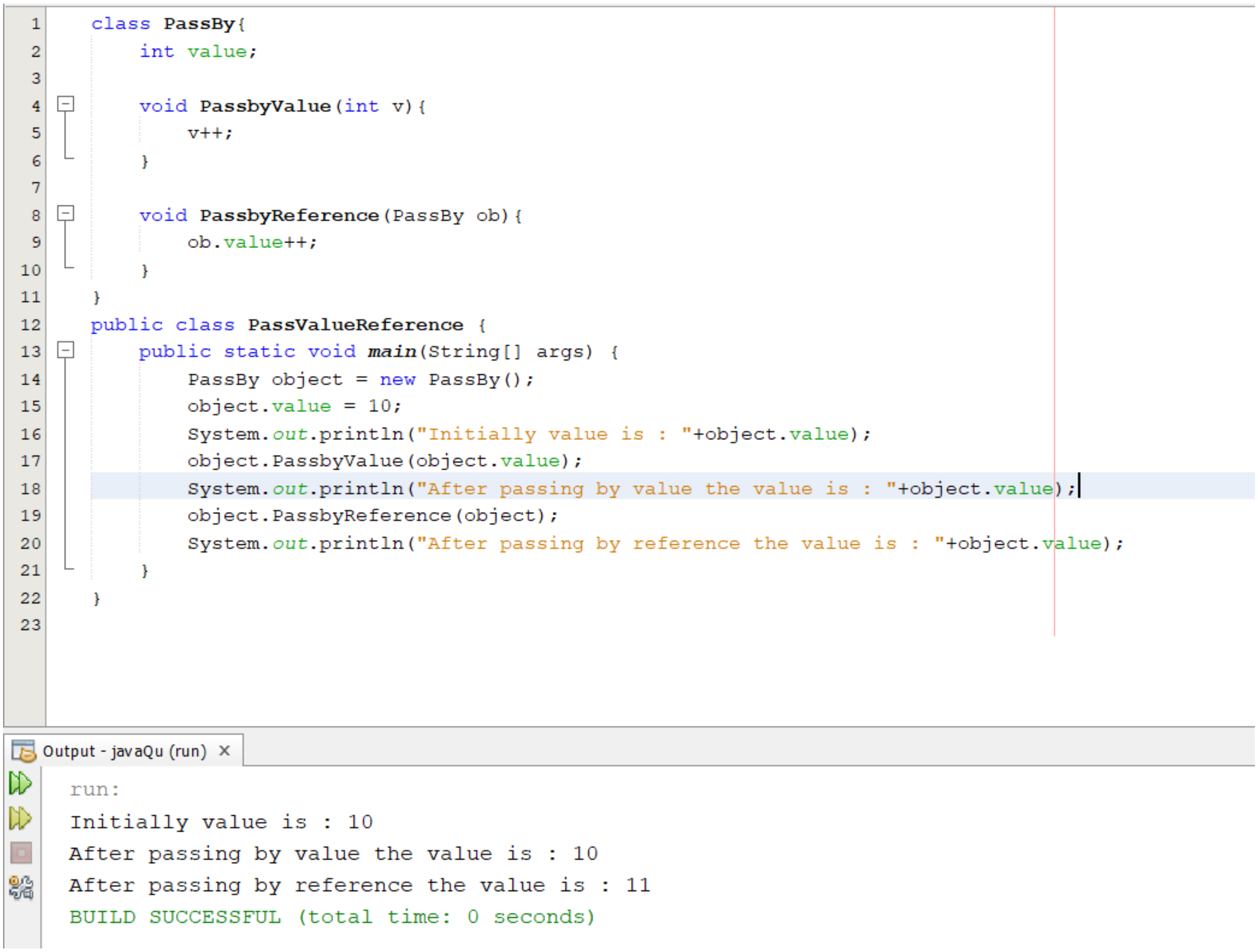
7. Why Java is called platform independent? Explain using JVM and compare it to "C".



Java is platform-independent because it does not depend on any platform. It uses a virtual machine. The Java programming language and all APIs are compiled into bytecodes. Bytecodes are effectively platform-independent. The virtual machine takes care of the differences between the bytecodes for the different platforms. The run-time requirements for Java are therefore very small. The Java virtual machine takes care of all hardware-related issues, so that no code has to be compiled for different hardware.

8. Explain pass-by-value and pass-by-reference with example.

Ans: Basically, pass-by-value means that the actual value of the variable is passed and pass-by-reference means the memory location is passed where the value of the variable is stored.



```
class PassBy{
    int value;

    void PassbyValue(int v){
        v++;
    }

    void PassbyReference(PassBy ob){
        ob.value++;
    }
}
public class PassValueReference {
    public static void main(String[] args) {
        PassBy object = new PassBy();
        object.value = 10;
        System.out.println("Initially value is : "+object.value);
        object.PassbyValue(object.value);
        System.out.println("After passing by value the value is : "+object.value);
        object.PassbyReference(object);
        System.out.println("After passing by reference the value is : "+object.value);
    }
}
```

```
run:
Initially value is : 10
After passing by value the value is : 10
After passing by reference the value is : 11
BUILD SUCCESSFUL (total time: 0 seconds)
```

9. You defined, for example a variable-“b”. Discuss it’s scope and lifetime with example.

Ans: A variable which is declared inside a class and outside all the methods and blocks is an instance variable. The general scope of an instance variable is throughout the class except in static methods. The lifetime of an instance variable is until the object stays in memory.

10. Define and initialize 5 dimensional int array.

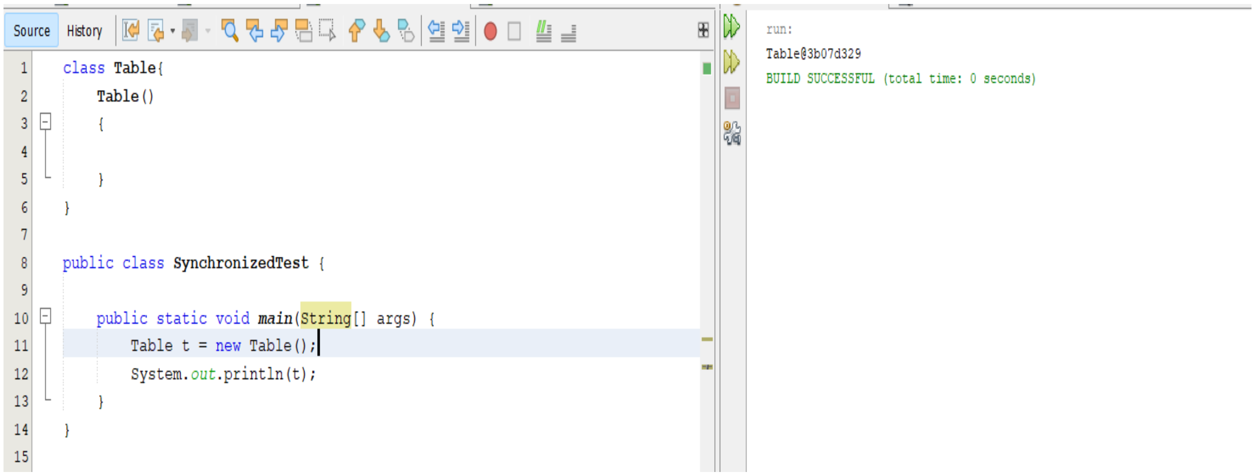
Ans: We can define multidimensional arrays in simple words as an array of arrays. Data in multidimensional arrays are stored in tabular form.

Int ar[][][][][] = new int[5][5][5][5][5];

11. Let there is a class – table. In the following sentence, what would be the output-

Table t = new Table();

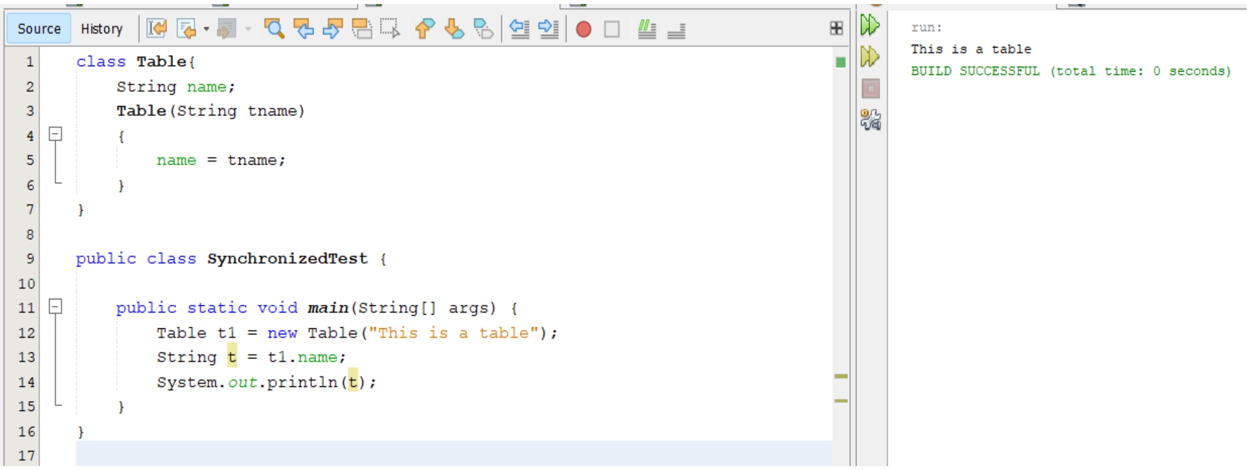
System.out.println(t);



```
class Table{
    Table()
    {

    }
}

public class SynchronizedTest {

    public static void main(String[] args) {
        Table t = new Table();
        System.out.println(t);
    }
}

```

```
run:
Table@3b07d329
BUILD SUCCESSFUL (total time: 0 seconds)
```

12. Now you want to print = “this is a table class” when we print “System.out.println(t)” statement earlier question without changing the print statement. How will you achieve that? Explain with code.



```
class Table{
    String name;
    Table(String tname)
    {
        name = tname;
    }
}

public class SynchronizedTest {

    public static void main(String[] args) {
        Table t1 = new Table("This is a table");
        String t = t1.name;
        System.out.println(t);
    }
}

```

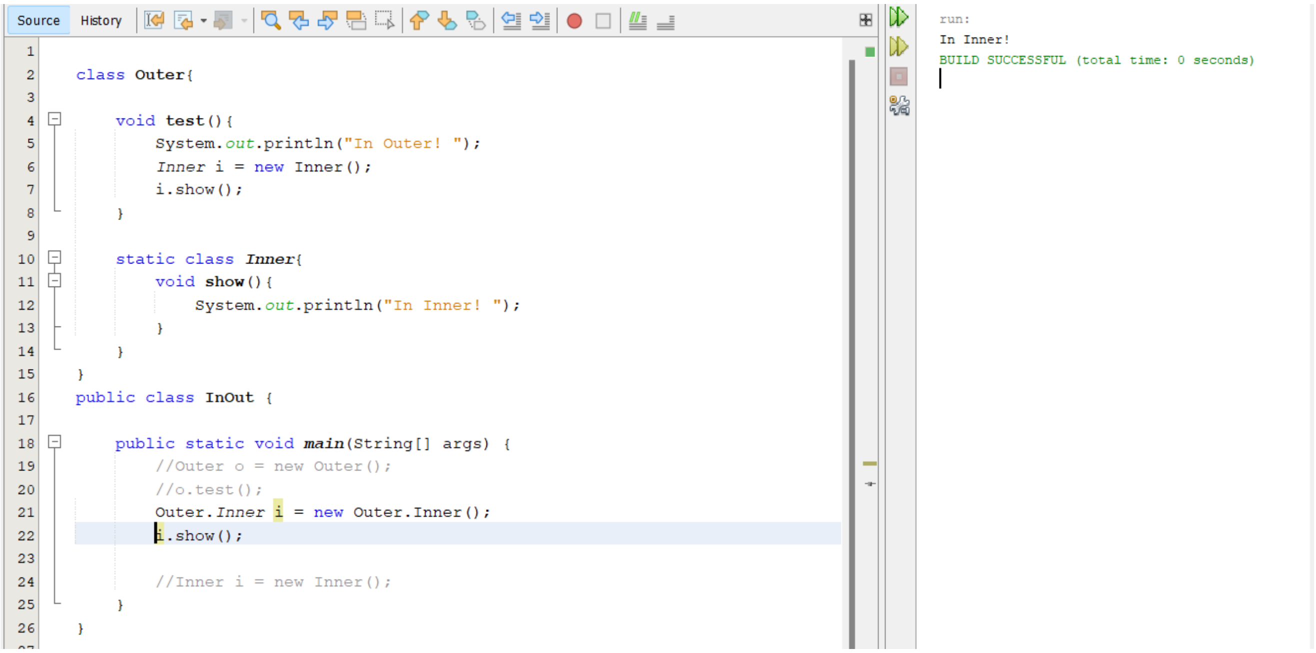
```
run:
This is a table
BUILD SUCCESSFUL (total time: 0 seconds)
```

13.Differentiate and explain with example – nested class and inner class.

Ans: When a class is defined within a scope of another class, then it becomes inner class. If the access modifier of the inner class is static, then it becomes nested class.

There are two types of nested classes: static and non-static. A static nested class is one that has the static modifier applied. Because it is static, it must access the non-static members of its enclosing class through an object. That is, it cannot refer to non-static members of its enclosing class directly. Because of this restriction, static nested classes are seldom used. The most important type of nested class is the inner class. An inner class is a non-static nested class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.

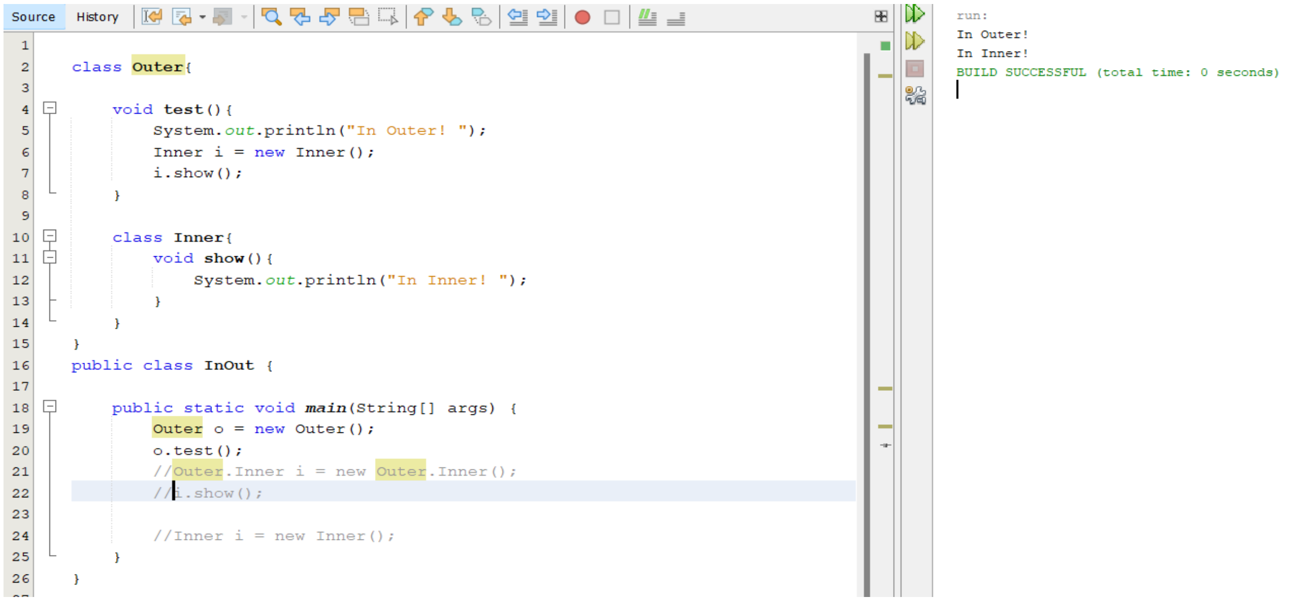
Nested class:



```

class Outer{

    void test(){
        System.out.println("In Outer! ");
        Inner i = new Inner();
        i.show();
    }

    static class Inner{
        void show(){
            System.out.println("In Inner! ");
        }
    }
}
public class InOut {

    public static void main(String[] args) {
        //Outer o = new Outer();
        //o.test();
        Outer.Inner i = new Outer.Inner();
        i.show();

        //Inner i = new Inner();
    }
}
```

```
run:
In Inner!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Inner class:



```

class Outer{

    void test(){
        System.out.println("In Outer! ");
        Inner i = new Inner();
        i.show();
    }

    class Inner{
        void show(){
            System.out.println("In Inner! ");
        }
    }
}
public class InOut {

    public static void main(String[] args) {
        Outer o = new Outer();
        o.test();
        //Outer.Inner i = new Outer.Inner();
        //i.show();

        //Inner i = new Inner();
    }
}
```

```
run:
In Outer!
In Inner!
BUILD SUCCESSFUL (total time: 0 seconds)
```

14. In light of three principles of OOP, explain with example method overloading, overriding. Concisely explain encapsulation with example.

Ans:

Java is a class-based object-oriented programming (OOP) language built around the concept of objects. OOP concepts are intended to improve code readability and reusability by defining how to structure your Java program efficiently. The core principles of object-oriented programming are:

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

1. Abstraction

Abstraction aims to hide complexity from users and show them only relevant information. For example, if you’re driving a car, you don’t need to know about its internal workings.

The same is true of Java classes. You can hide internal implementation details using abstract classes or interfaces. On the abstract level, you only need to define the method signatures (name and parameter list) and let each class implement them in their own way.

Abstraction in Java:

- Hides the underlying complexity of data
- Helps avoid repetitive code
- Presents only the signature of internal functionality
- Gives flexibility to programmers to change the implementation of abstract behavior
- Partial abstraction (0-100%) can be achieved with abstract classes
- Total abstraction (100%) can be achieved with interfaces

2. Encapsulation

Encapsulation helps with data security, allowing you to protect the data stored in a class from system-wide access. As the name suggests, it safeguards the internal contents of a class like a capsule.

You can implement encapsulation in Java by making the fields (class variables) private and accessing them via their public getter and setter methods. JavaBeans are examples of fully encapsulated classes.

Encapsulation in Java:

- Restricts direct access to data members (fields) of a class
- Fields are set to private
- Each field has a getter and setter method
- Getter methods return the field
- Setter methods let us change the value of the field

3. Inheritance

Inheritance makes it possible to create a child class that inherits the fields and methods of the parent class. The child class can override the values and methods of the parent class, but it's not necessary. It can also add new data and functionality to its parent.

Parent classes are also called superclasses or base classes, while child classes are known as subclasses or derived classes as well. Java uses the extends keyword to implement the principle of inheritance in code.

Inheritance in Java:

- A class (child class) can extend another class (parent class) by inheriting its features
- Implements the DRY (Don't Repeat Yourself) programming principle
- Improves code reusability
- Multi-level inheritance is allowed in Java (a child class can have its own child class as well)
- Multiple inheritances are not allowed in Java (a class can't extend more than one class)

4. Polymorphism

Polymorphism refers to the ability to perform a certain action in different ways. In Java, polymorphism can take two forms: method overloading and method overriding.

Method overloading happens when various methods with the same name are present in a class. When they are called, they are differentiated by the number, order, or types of their parameters. Method overriding occurs when a child class overrides a method of its parent.

Polymorphism in Java:

- The same method name is used several times
- Different methods of the same name can be called from an object
- All Java objects can be considered polymorphic (at the minimum, they are of their own type and instances of the Object class)
- Static polymorphism in Java is implemented by method overloading
- Dynamic polymorphism in Java is implemented by method overriding

Method overloading and method overriding – question 46

Encapsulation with example – question 30 & 38

15. How would you pause a java program for a while?

Ans: To pause the execution of a thread, we use "sleep()" method of Thread class.

Syntax: Thread.currentThread().sleep(miliseconds).

Example: Thread.currentThread().sleep(100);

16. Which class would you inherit to make a java application program?

Ans: Don’t know… may be Jframe..

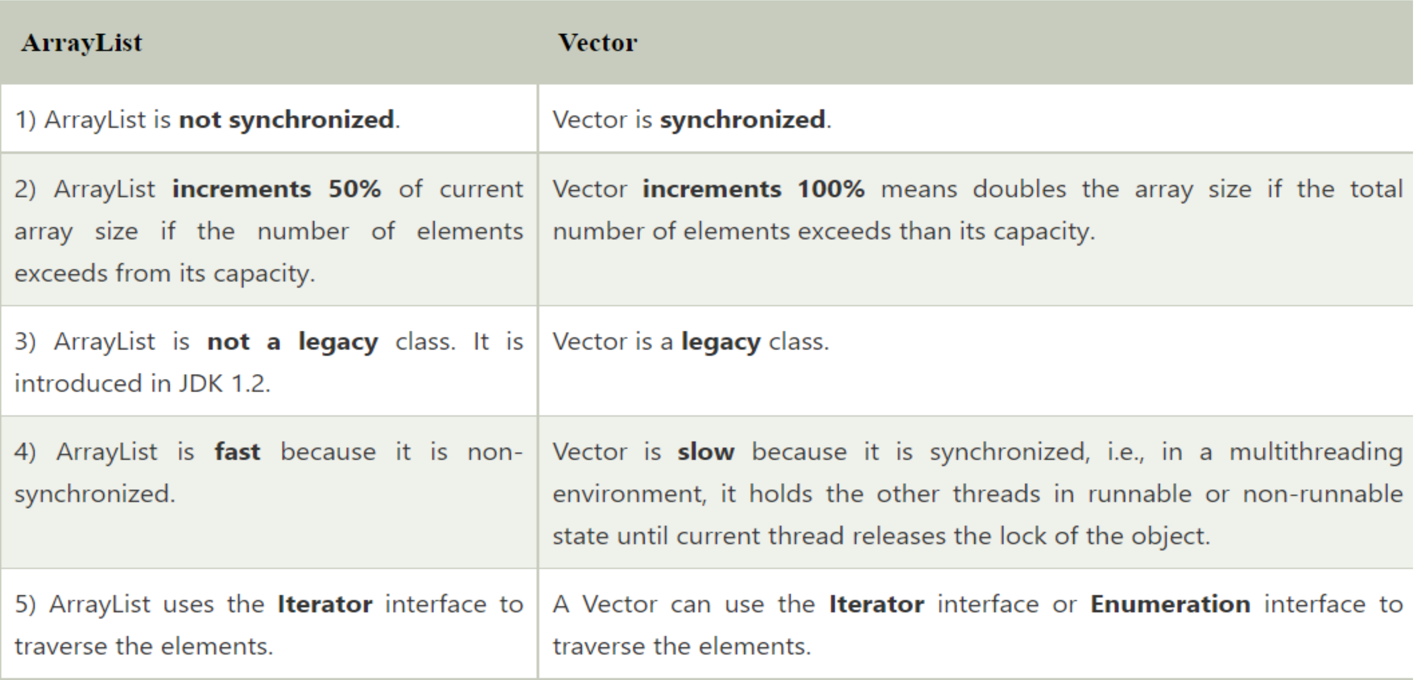
17.Which method is used to register a mouse motion listener?

Ans: addMouseMotionListener() method is used to register a mouse motion listener.

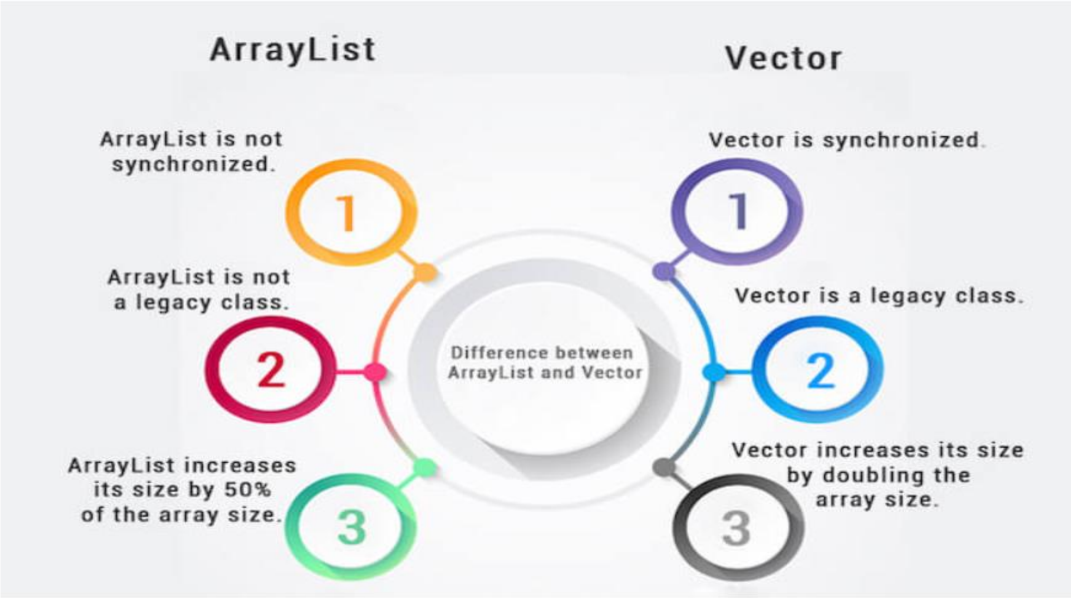
18. Differentiate Vector vs ArrayList.

Ans: ArrayList and Vectors both implement the List interface, and both use **(dynamically resizable) arrays** for their internal data structure, much like using an ordinary array.

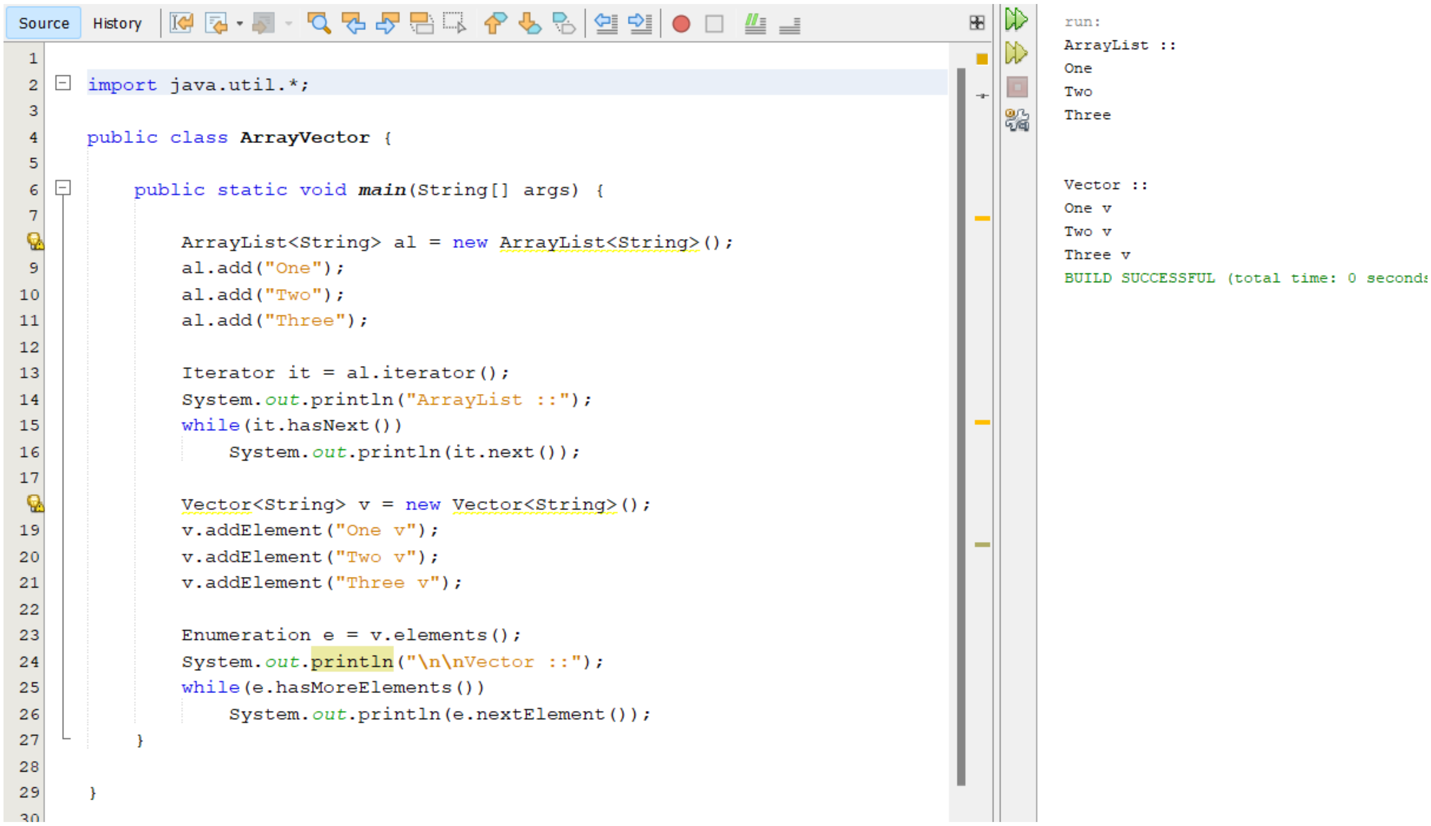
ArrayList and Vector both implements List interface and maintains insertion order.



ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity.	Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
5) ArrayList uses the **Iterator** interface to traverse the elements.	A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements.



ArrayList
Vector
ArrayList is not synchronized.
1
Vector is synchronized.
1
ArrayList is not a legacy class.
2
Difference between ArrayList and Vector
Vector is a legacy class.
2
ArrayList increases its size by 50% of the array size.
3
Vector increases its size by doubling the array size.
3



```java
import java.util.*;

public class ArrayVector {

    public static void main(String[] args) {

        ArrayList<String> al = new ArrayList<String>();
        al.add("One");
        al.add("Two");
        al.add("Three");

        Iterator it = al.iterator();
        System.out.println("ArrayList ::");
        while(it.hasNext())
            System.out.println(it.next());

        Vector<String> v = new Vector<String>();
        v.addElement("One v");
        v.addElement("Two v");
        v.addElement("Three v");

        Enumeration e = v.elements();
        System.out.println("\n\nVector ::");
        while(e.hasMoreElements())
            System.out.println(e.nextElement());
    }

}
```

```
run:
ArrayList ::
One
Two
Three


Vector ::
One v
Two v
Three v
BUILD SUCCESSFUL (total time: 0 seconds
```

19. Differentiate primitive data type vs rest of the data types.

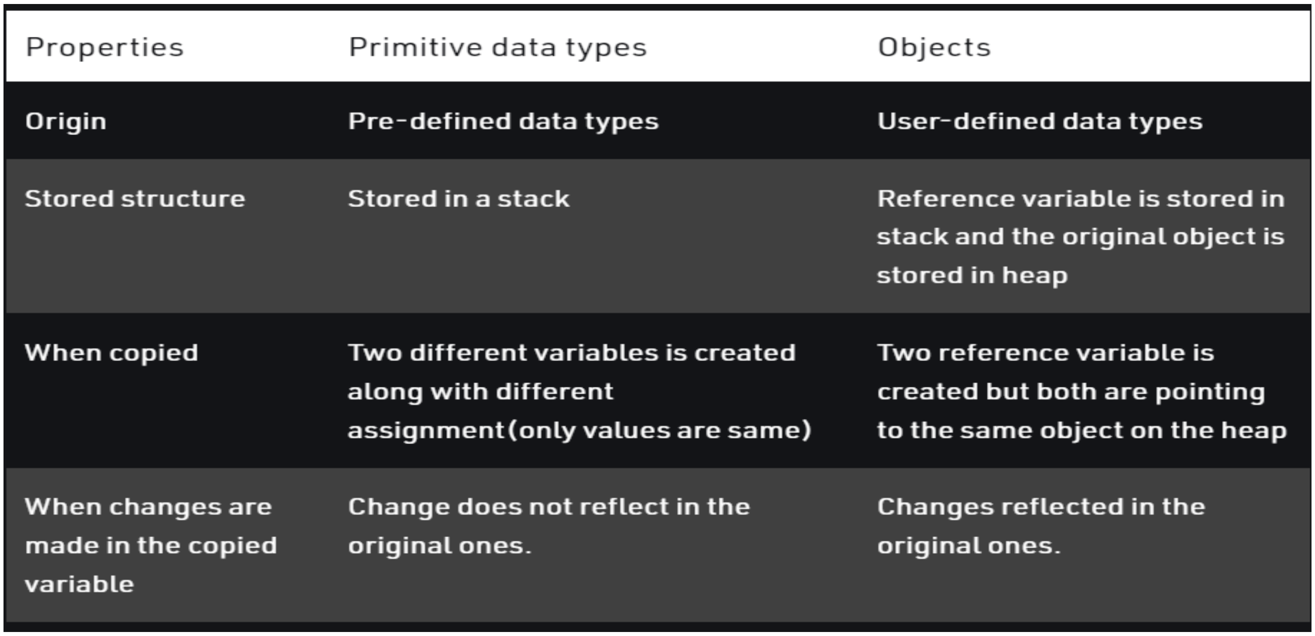
Ans:

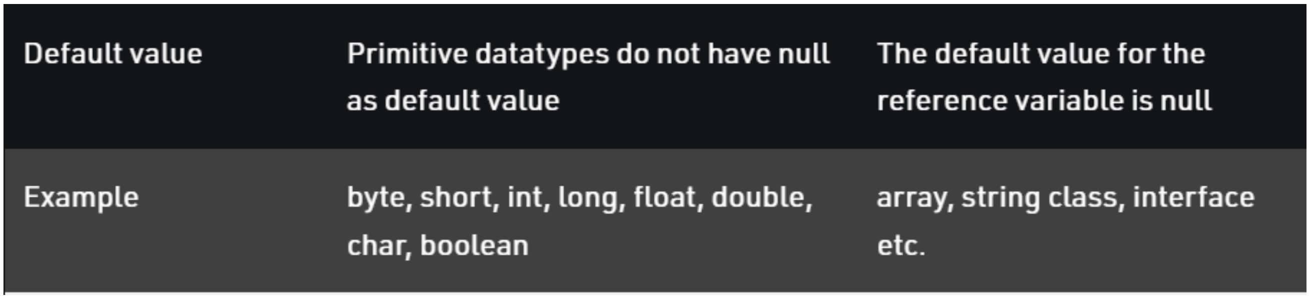
Primitive Data Type**:**

In Java, the primitive data types are the predefined data types of Javas. They specify the size and type of any standard values. Java has 8 primitive data types namely byte, short, int, long, float, double, char and boolean. When a primitive data type is stored, it is the stack that the values will be assigned. When a variable is copied then another copy of the variable is created and changes made to the copied variable will not reflect changes in the original variable. Here is a Java program to demonstrate all the primitive data types in Java.

Object Data Type:

These are also referred to as Non-primitive or Reference Data Type. They are so-called because they refer to any particular objects. Unlike the primitive data types, the non-primitive ones are created by the users in Java. Examples include arrays, strings, classes, interfaces etc. When the reference variables will be stored, the variable will be stored in the stack and the original object will be stored in the heap. In Object data type although two copies will be created, they both will point to the same variable in the heap, hence changes made to any variable will reflect the change in both the variables. Here is a Java program to demonstrate arrays (an object data type) in Java.





Properties	Primitive data types	Objects
Origin	Pre-defined data types	User-defined data types
Stored structure	Stored in a stack	Reference variable is stored in stack and the original object is stored in heap
When copied	Two different variables is created along with different assignment(only values are same)	Two reference variable is created but both are pointing to the same object on the heap
When changes are made in the copied variable	Change does not reflect in the original ones.	Changes reflected in the original ones.
Default value	Primitive datatypes do not have null as default value	The default value for the reference variable is null
Example	byte, short, int, long, float, double, char, boolean	array, string class, interface etc.

20.Why do you override constructor?

Ans: Overriding occurs when a subclass (child class) has the same method as the parent class.

Constructor is a block of code that allows you to create an object of class and has same name as class with no explicit return type.

Whenever a class (child class) extends another class (parent class), the sub class inherits state and behavior in the form of variables and methods from its super class but it does not inherit constructor of super class because Constructors are special and have same name as class name. So, if constructors were inherited in child class, then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name. If we define Parent class constructor inside Child class it will give compile time error for return type and consider it a method.

21. Show the auto type-promotion rules in java.

Ans: There are several type promotion rules in Java that are followed while evaluating expressions-

- All byte, short and char values are promoted to int.
- If any operand is long then the expression result is long. i.e. whole expression is promoted to long.
- If any operand is a float then the expression result is float. i.e. whole expression is automatically promoted to float.

- If any operand is a double then the expression result is double. i.e. whole expression is promoted to double in Java.

22. Differentiate Set vs Map.

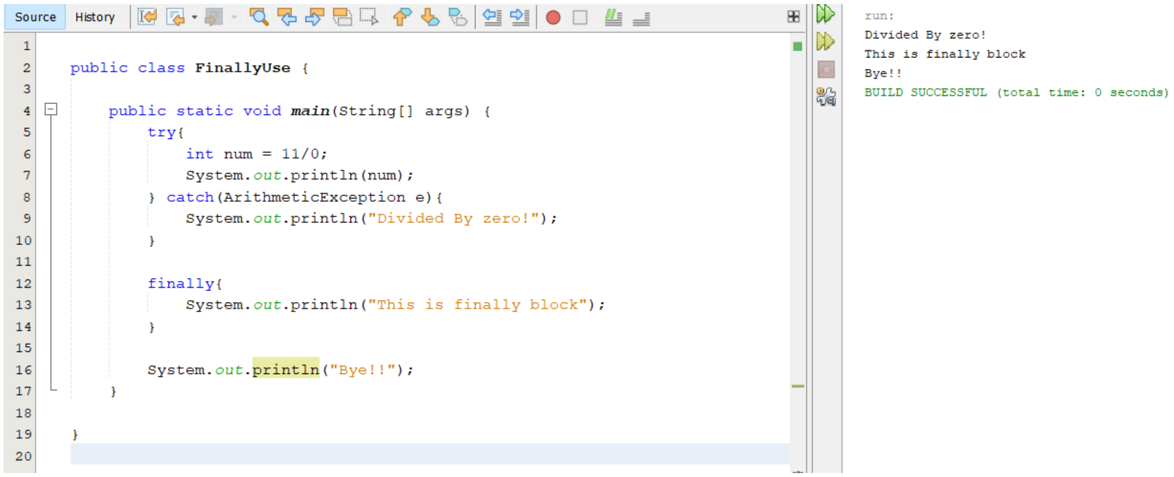
Ans: Both interfaces are used to store the collection of objects as a single unit. The main difference between Set and Map is that Set contains only data elements, and the Map contains the data in the key-value pair, so Map contains key and its value.

Now, let's understand some major differences between both of them.

S.No.	Set	Map
1.	Set is used to construct the mathematical Set in Java.	Map is used to do mapping in the database.
2.	It cannot contain repeated values.	It can have the same value for different keys.
3.	Set doesn't allow us to add the same elements in it. Each class that implements the Set interface contains only the unique value.	Map contains unique key and repeated values. In Map, one or more keys can have the same values, but two keys cannot be the same.
4.	We can easily iterate the Set elements using the keyset() and the entryset() method of it.	Map elements cannot be iterated. We need to convert Map into Set for iterating the elements.
5.	Insertion order is not maintained by the Set interface. However, some of its classes, like LinkedHashSet, maintains the insertion order.	The insertion order is also not maintained by the Map. However, some of the Map classes like TreeMap and LinkedHashMap does the same.

23. Why do we use finally in exception handling?

Ans: A **finally block** contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

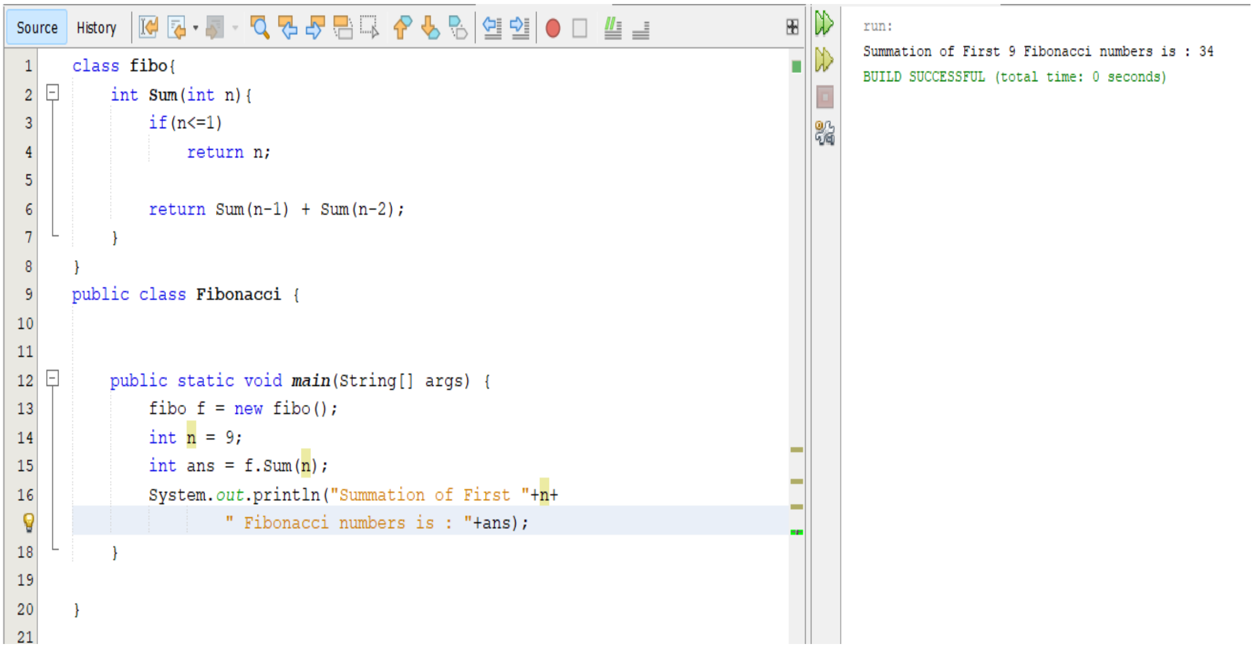


```

public class FinallyUse {

    public static void main(String[] args) {
        try{
            int num = 11/0;
            System.out.println(num);
        } catch(ArithmeticException e){
            System.out.println("Divided By zero!");
        }

        finally{
            System.out.println("This is finally block");
        }

        System.out.println("Bye!!");
    }

}

```

```
run:
Divided By zero!
This is finally block
Bye!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

24. Write a recursive method Sum (as a class member) to sum the first N Fibonacci numbers.

Ans:



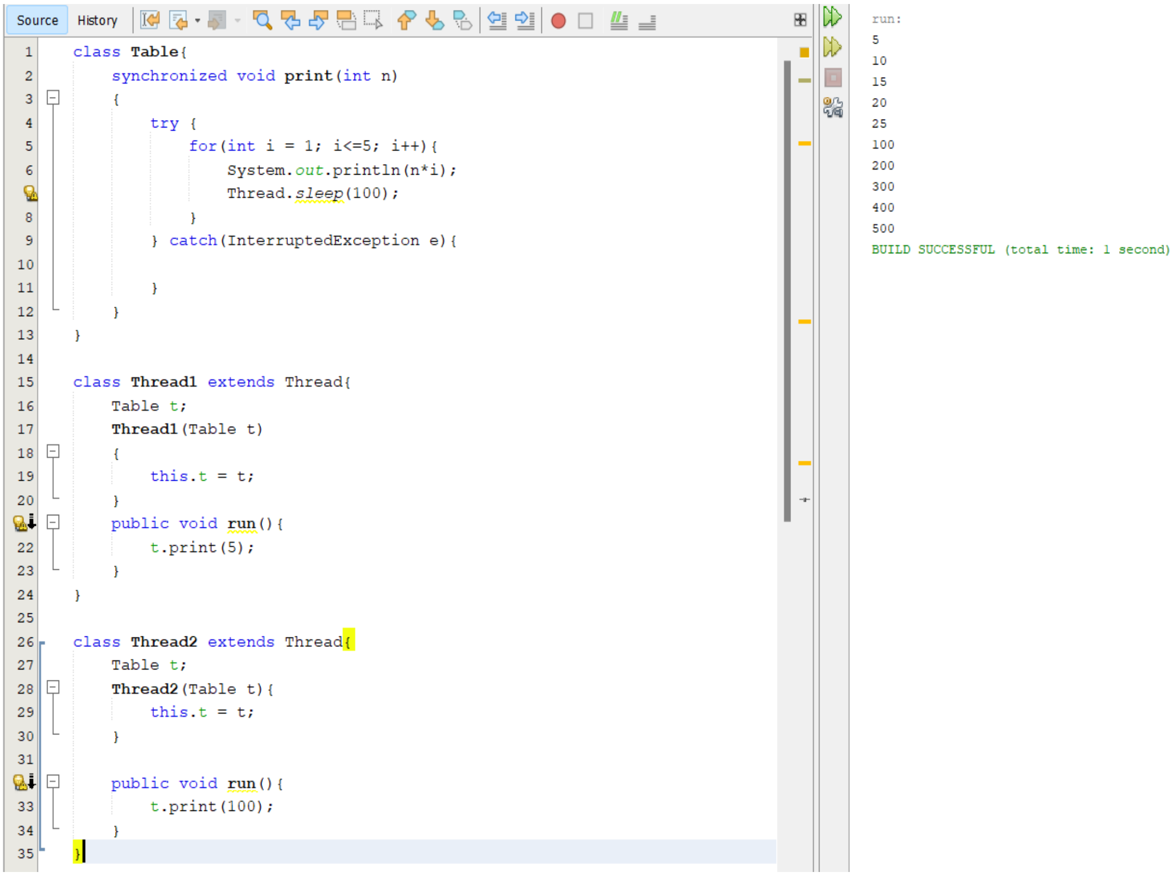
```
class fibo{
    int Sum(int n){
        if(n<=1)
            return n;

        return Sum(n-1) + Sum(n-2);
    }
}
public class Fibonacci {


    public static void main(String[] args) {
        fibo f = new fibo();
        int n = 9;
        int ans = f.Sum(n);
        System.out.println("Summation of First "+n+
                " Fibonacci numbers is : "+ans);
    }

}
```

```
run:
Summation of First 9 Fibonacci numbers is : 34
BUILD SUCCESSFUL (total time: 0 seconds)
```

25. What is thread synchronization? How would you achieve that?

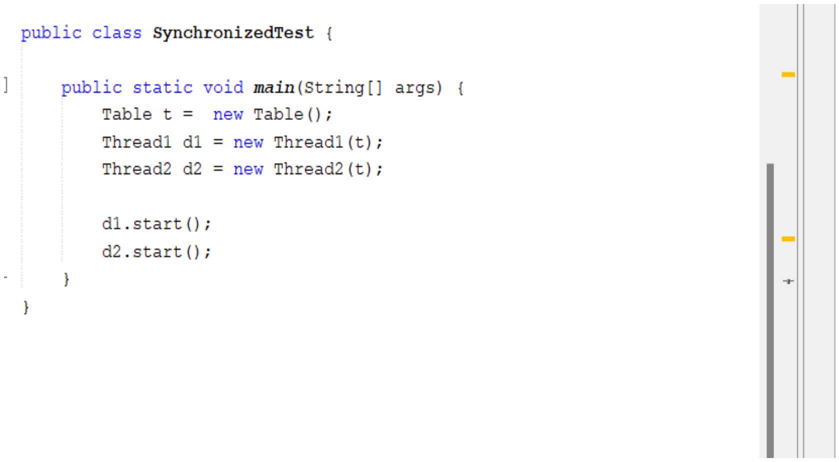
Ans: Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.



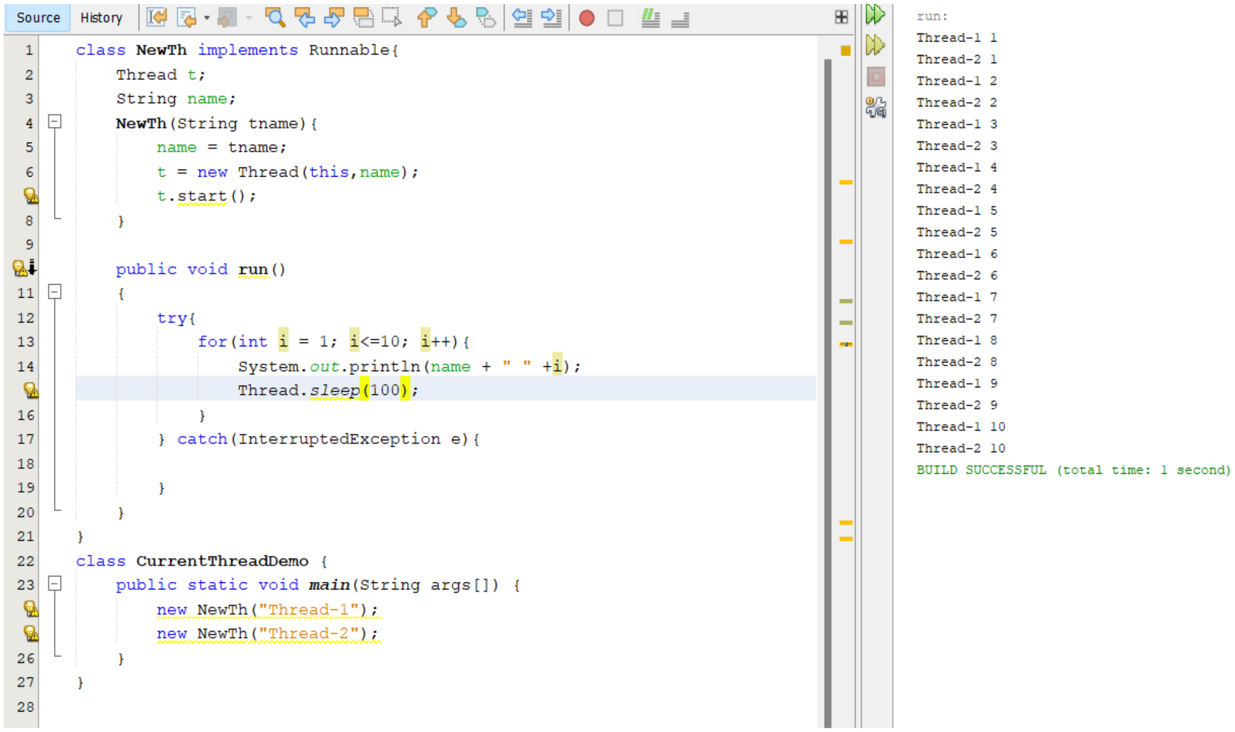
```
class Table{
    synchronized void print(int n)
    {
        try {
            for(int i = 1; i<=5; i++){
                System.out.println(n*i);
                Thread.sleep(100);
            }
        } catch(InterruptedException e){

        }
    }
}

class Thread1 extends Thread{
    Table t;
    Thread1(Table t)
    {
        this.t = t;
    }
    public void run(){
        t.print(5);
    }
}

class Thread2 extends Thread{
    Table t;
    Thread2(Table t){
        this.t = t;
    }

    public void run(){
        t.print(100);
    }
}
```

```
run:
5
10
15
20
25
100
200
300
400
500
BUILD SUCCESSFUL (total time: 1 second)
```



```
public class SynchronizedTest {

    public static void main(String[] args) {
        Table t =  new Table();
        Thread1 d1 = new Thread1(t);
        Thread2 d2 = new Thread2(t);

        d1.start();
        d2.start();
    }
}
```

26. Create simple 2 thread program that prints 1 to 10 maintaining following structure- “Thread-1 1”, “Thread-2 1” etc.



```
class NewTh implements Runnable{
    Thread t;
    String name;
    NewTh(String tname){
        name = tname;
        t = new Thread(this,name);
        t.start();
    }

    public void run()
    {
        try{
            for(int i = 1; i<=10; i++){
                System.out.println(name + " " +i);
                Thread.sleep(100);
            }
        } catch(InterruptedException e){

        }
    }
}
class CurrentThreadDemo {
    public static void main(String args[]) {
        new NewTh("Thread-1");
        new NewTh("Thread-2");
    }
}
```

```
run:
Thread-1 1
Thread-2 1
Thread-1 2
Thread-2 2
Thread-1 3
Thread-2 3
Thread-1 4
Thread-2 4
Thread-1 5
Thread-2 5
Thread-1 6
Thread-2 6
Thread-1 7
Thread-2 7
Thread-1 8
Thread-2 8
Thread-1 9
Thread-2 9
Thread-1 10
Thread-2 10
BUILD SUCCESSFUL (total time: 1 second)
```

27. Explain wait(), notify(), notifyAll() with example.

Ans:

wait() -> Waits on another thread of execution.

notify() -> Resumes execution of a thread waiting on the invoking object.

notifyAll() -> Resumes execution of all threads waiting on the invoking object.

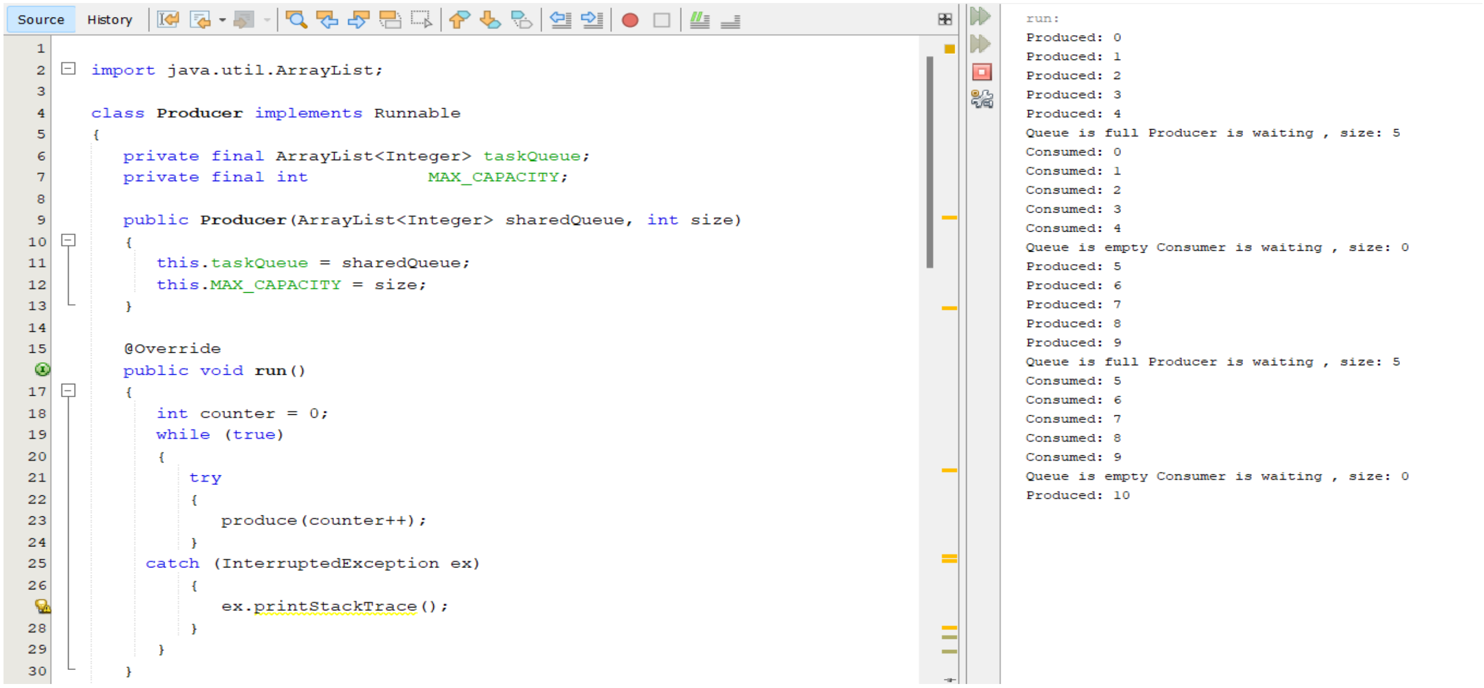
To avoid polling, Java includes an elegant interprocess communication mechanism via the wait( ), notify( ), and notifyAll( ) methods. These methods are implemented as final methods in Object, so all classes have them. All three methods can be called only from within a synchronized context. Although conceptually advanced from a computer science perspective, the rules for using these methods are actually quite simple:

- wait( ) tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ) or notifyAll( ).

- notify( ) wakes up a thread that called wait( ) on the same object.

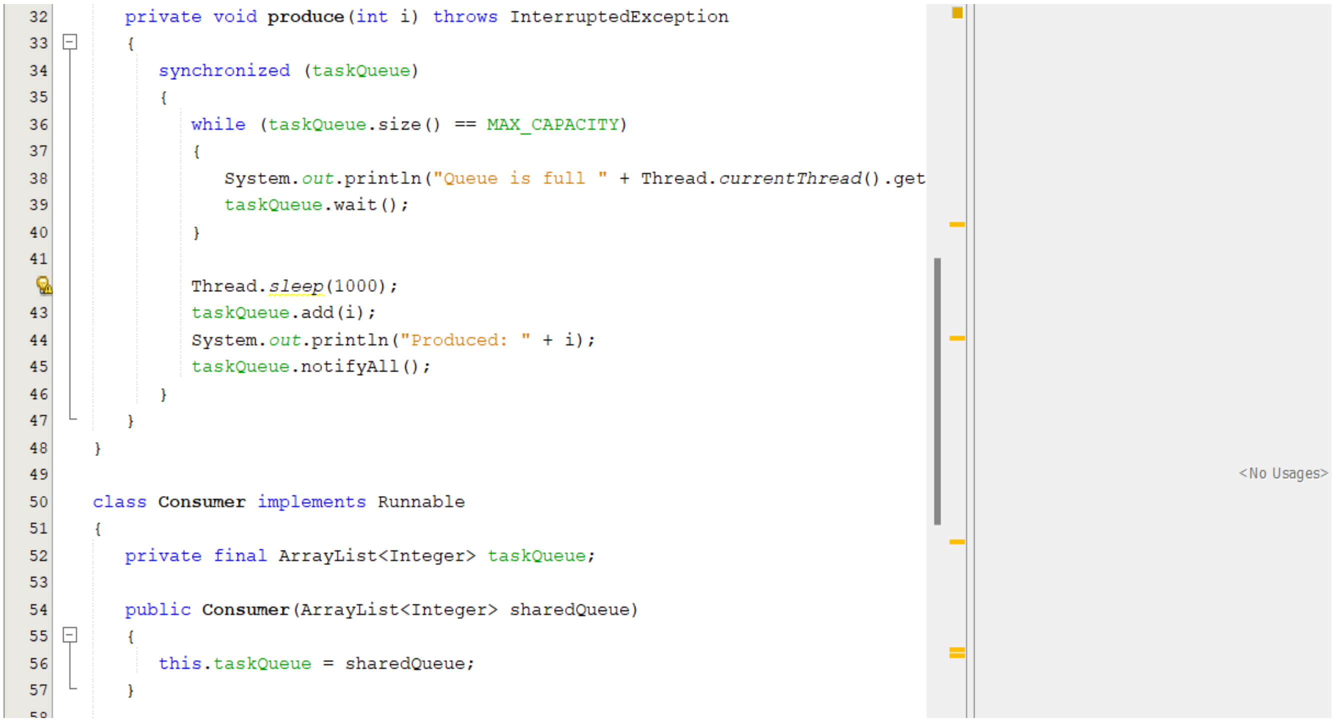
- notifyAll( ) wakes up all the threads that called wait( ) on the same object. One of the threads will be granted access.

Example:

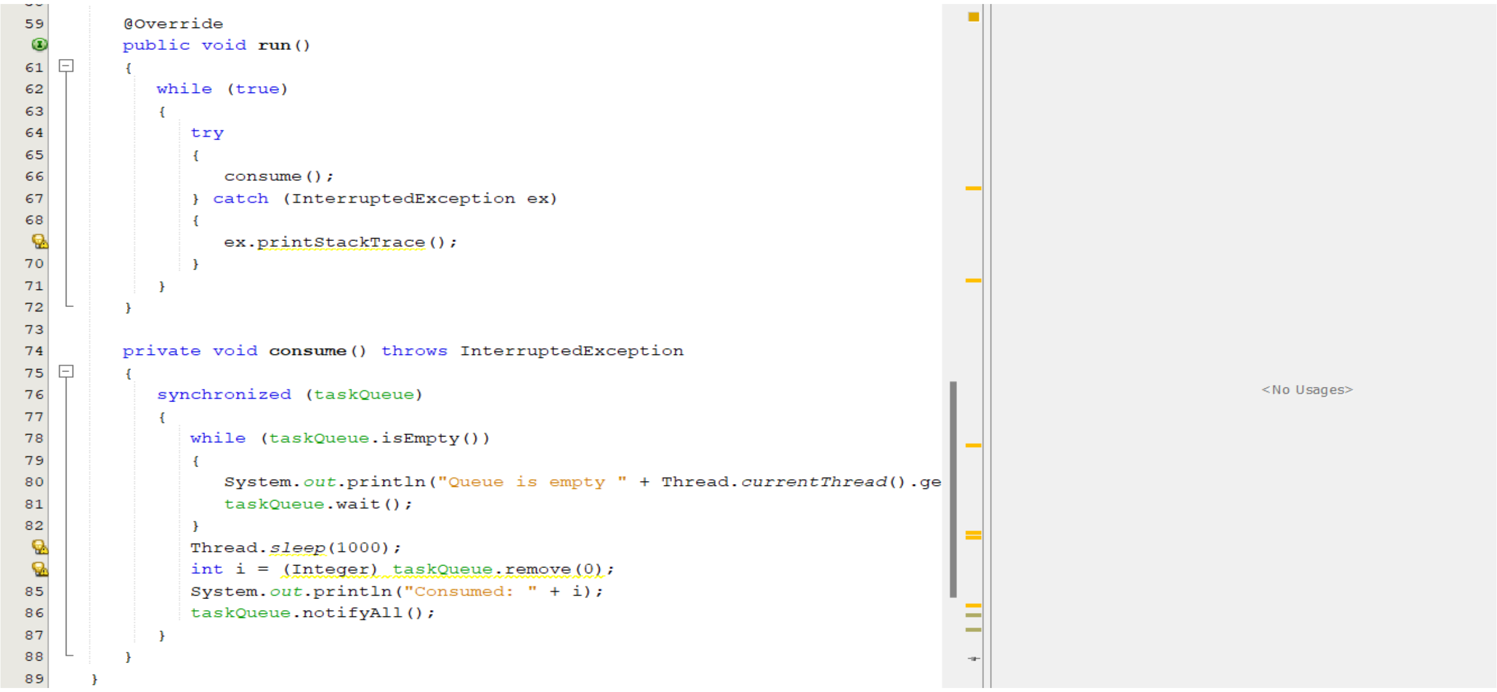


```

import java.util.ArrayList;

class Producer implements Runnable
{
    private final ArrayList<Integer> taskQueue;
    private final int           MAX_CAPACITY;

    public Producer(ArrayList<Integer> sharedQueue, int size)
    {
        this.taskQueue = sharedQueue;
        this.MAX_CAPACITY = size;
    }

    @Override
    public void run()
    {
        int counter = 0;
        while (true)
        {
            try
            {
                produce(counter++);
            }
        catch (InterruptedException ex)
            {
                ex.printStackTrace();
            }
        }
    }
```

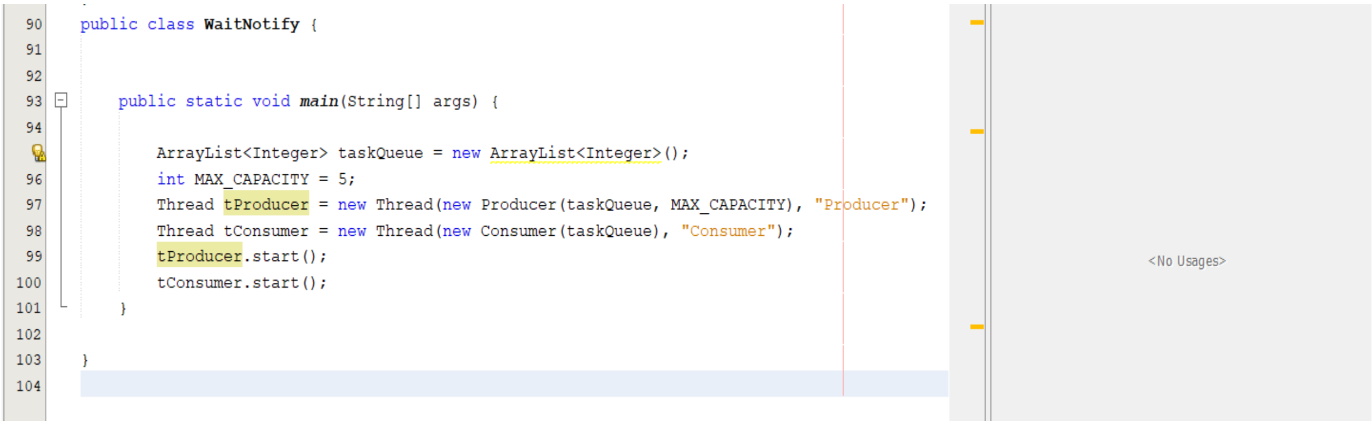
```
run:
Produced: 0
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Queue is full Producer is waiting , size: 5
Consumed: 0
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4
Queue is empty Consumer is waiting , size: 0
Produced: 5
Produced: 6
Produced: 7
Produced: 8
Produced: 9
Queue is full Producer is waiting , size: 5
Consumed: 5
Consumed: 6
Consumed: 7
Consumed: 8
Consumed: 9
Queue is empty Consumer is waiting , size: 0
Produced: 10
```



```
    private void produce(int i) throws InterruptedException
    {
        synchronized (taskQueue)
        {
            while (taskQueue.size() == MAX_CAPACITY)
            {
                System.out.println("Queue is full " + Thread.currentThread().get
                taskQueue.wait();
            }

            Thread.sleep(1000);
            taskQueue.add(i);
            System.out.println("Produced: " + i);
            taskQueue.notifyAll();
        }
    }
}

class Consumer implements Runnable
{
    private final ArrayList<Integer> taskQueue;

    public Consumer(ArrayList<Integer> sharedQueue)
    {
        this.taskQueue = sharedQueue;
    }
```



```
    @Override
    public void run()
    {
        while (true)
        {
            try
            {
                consume();
            } catch (InterruptedException ex)
            {
                ex.printStackTrace();
            }
        }
    }

    private void consume() throws InterruptedException
    {
        synchronized (taskQueue)
        {
            while (taskQueue.isEmpty())
            {
                System.out.println("Queue is empty " + Thread.currentThread().ge
                taskQueue.wait();
            }
            Thread.sleep(1000);
            int i = (Integer) taskQueue.remove(0);
            System.out.println("Consumed: " + i);
            taskQueue.notifyAll();
        }
    }
}
```



```
public class WaitNotify {


    public static void main(String[] args) {

        ArrayList<Integer> taskQueue = new ArrayList<Integer>();
        int MAX_CAPACITY = 5;
        Thread tProducer = new Thread(new Producer(taskQueue, MAX_CAPACITY), "Producer");
        Thread tConsumer = new Thread(new Consumer(taskQueue), "Consumer");
        tProducer.start();
        tConsumer.start();
    }

}
```

28. There are two types of input streams in JAVA. Explain those.

Ans: Java defines two types of streams:

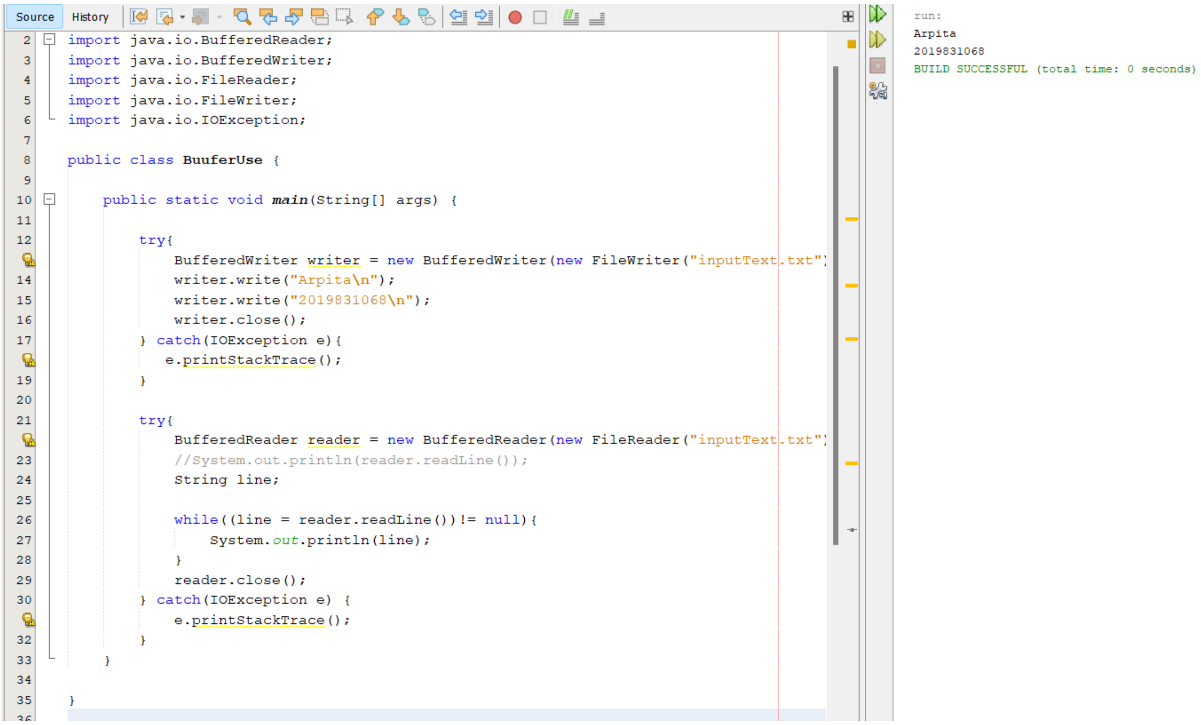
Byte streams and Character streams.

- Byte streams provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary data.
- Character streams provide a convenient means for handling input and output of characters. They use Unicode and, therefore, can be internationalized. Also, in some cases, character streams are more efficient than byte streams.

29. What is buffer? Explain and show use of buffer in code.

Ans: It is the block of memory into which we can write data, which we can later be read again. The memory block is wrapped with a NIO buffer object, which provides easier methods to work with the memory block.

A **Buffer** is a portion in the memory that is used to store a **stream** of data from peripheral devices. Then from this buffer this **stream** of data is collected and stored in variables. A stream can be defined as a continuous flow of data. The buffer is quite useful as Java deals everything as a String.



```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class BuuferUse {

    public static void main(String[] args) {

        try{
            BufferedWriter writer = new BufferedWriter(new FileWriter("inputText.txt")
            writer.write("Arpita\n");
            writer.write("2019831068\n");
            writer.close();
        } catch(IOException e){
           e.printStackTrace();
        }

        try{
            BufferedReader reader = new BufferedReader(new FileReader("inputText.txt")
            //System.out.println(reader.readLine());
            String line;

            while((line = reader.readLine())!= null){
                System.out.println(line);
            }
            reader.close();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }

}
```

```
run:
Arpita
2019831068
BUILD SUCCESSFUL (total time: 0 seconds)
```

30. Differentiate and explain different access modifiers with example.

Ans:

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

1) Private

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

Note: A class cannot be private or protected except nested class.

2) Default

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```java
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
```

```java
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

```
Output:Hello
```

4) Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

```
Output:Hello
```

Java Access Modifiers with Method Overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```
class A{
protected void msg(){System.out.println("Hello java");}
}

public class Simple extends A{
void msg(){System.out.println("Hello java");}//C.T.Error
 public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();
   }
}
```

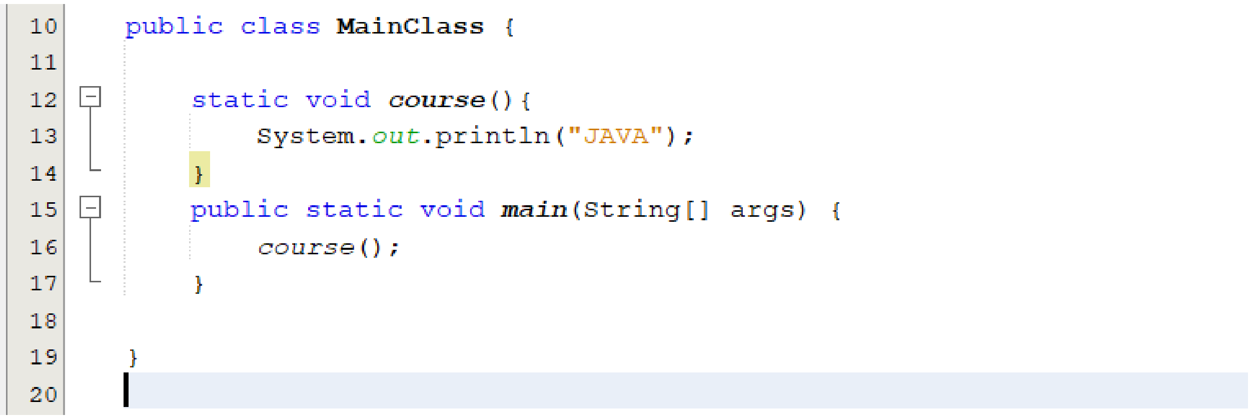
The default modifier is more restrictive than protected. That is why, there is a compile-time error.

31. Notice the code below. Is there any problem? If it is, correct the code and explain.

```
public class MainClass {
        void course() {
                System.out.println(“JAVA”);
        }
        public static void main(String[] args) {
                course();
        }
}
```

Ans: Here the problem is course() method is a non-static method. Non-static method course can-not be referenced from a static context.

So, the solution is:



```
public class MainClass {

    static void course(){
        System.out.println("JAVA");
    }
    public static void main(String[] args) {
        course();
    }


}

```

32.Can you override a private or static method in java? Explain.

Ans: No, we cannot override private or static methods in Java.

Private methods in Java are not visible to any other class which limits their scope to the class in which they are declared.

No, you **cannot override private method**, hence the method is called private so that no class extending that class has any access to the **private** method. Private methods are not visible to child classes.

Static methods also cannot be overridden, because static methods are a part of the Class itself, and not a part of any instance(object) of that class. You however can declare same static method with same signature in child classes, but that would not be considered as **runtime polymorphism** (override of methods).

NOTE: Static methods can’t be overridden because methods are overridden at run time. Static methods are associated with classes while instance methods are associated with objects. So in Java, the main() method also can’t be overridden.

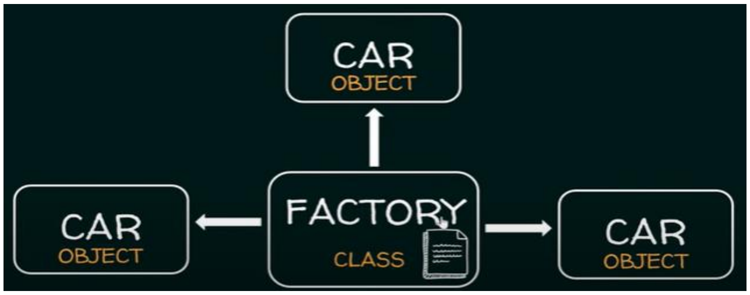
NOTE: Constructors can be overloaded but not overridden.

33.What do you mean by 'Object' in oop. Dissect and explain 'Car' object with a diagram.

Ans: In OOP, objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process.

What is class? -A blueprint to create objects

What is Object? - An instance of a class



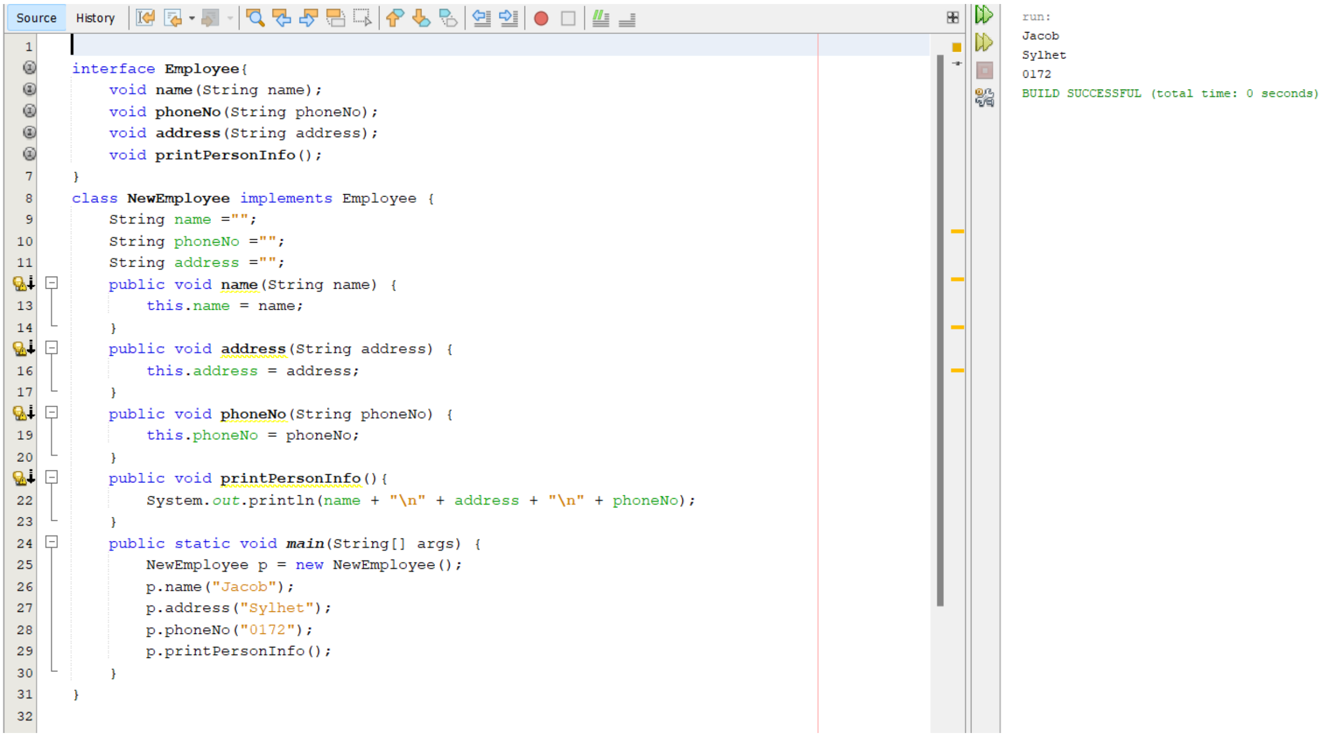
Suppose that we have a factory and this factory produces cars.

In order to be able to produce a car we should have some characteristics or properties for each car. So, this factory contains the characteristics or properties of a car. For example, the color of the car, the model and other information. So, in this case, the factory is the class, and each car will be an object. So, A class is a blueprint to create objects. It has the properties of each object. And we can create objects from this class. Moreover, each object is an instance of a class.

34.Notice and determine if there are any problem with the code below. If there is, correct and explain.

```
interface Employee{
  void name(String name);
  void phoneNo(String phoneNo);
}

class NewEmployee implements Employee{
  String name = "";
  String phoneNo = "";

  public void name(String name) {
    this.name = name;
  }

  void printPersonInfo(){
    System.out.println(name + "\n" + address + "\n" + phoneNo);
  }

public static void main(String[] args) {
      Person p = new NewPerson();
      p.name("Jacob");
      p.printPersonInfo();
  }
}
```

Ans:



```

interface Employee{
    void name(String name);
    void phoneNo(String phoneNo);
    void address(String address);
    void printPersonInfo();
}
class NewEmployee implements Employee {
    String name ="";
    String phoneNo ="";
    String address ="";
    public void name(String name) {
        this.name = name;
    }
    public void address(String address) {
        this.address = address;
    }
    public void phoneNo(String phoneNo) {
        this.phoneNo = phoneNo;
    }
    public void printPersonInfo(){
        System.out.println(name + "\n" + address + "\n" + phoneNo);
    }
    public static void main(String[] args) {
        NewEmployee p = new NewEmployee();
        p.name("Jacob");
        p.address("Sylhet");
        p.phoneNo("0172");
        p.printPersonInfo();
    }
}

```

```
run:
Jacob
Sylhet
0172
BUILD SUCCESSFUL (total time: 0 seconds)
```

35.What is meant by “this” in java?

Ans: The “this” keyword refers to the current object in a method or constructor.

The most common use of the “this” keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

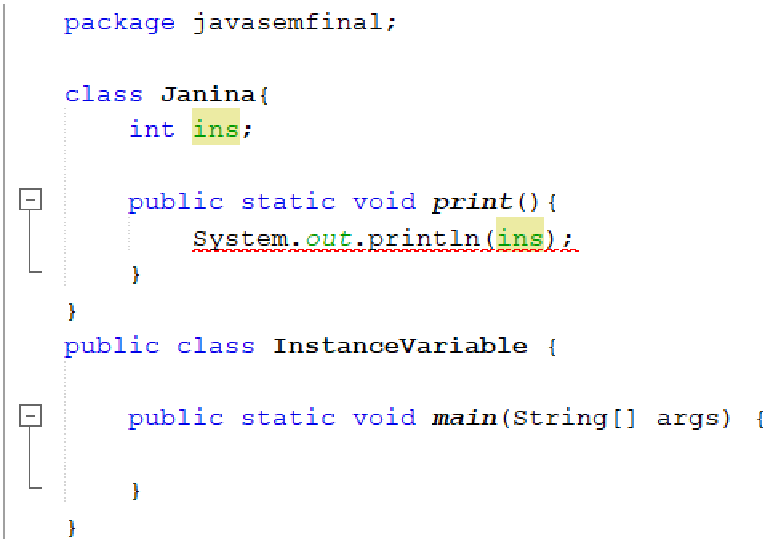
this can also be used to:

- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

** Interface abstract method cannot have a body;

36.Let's assume we defined a variable 'test'. Explain its scope and lifetime with example.

Ans: A variable which is declared inside a class and outside all the methods and blocks is an instance variable. The general scope of an instance variable is throughout the class except in static methods. The lifetime of an instance variable is until the object stays in memory.



```
package javasemfinal;

class Janina{
    int ins;

    public static void print(){
        System.out.println(ins);
    }
}
public class InstanceVariable {

    public static void main(String[] args) {

    }
}
```

37.Explain the keyword "Super" in java.

Ans: The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

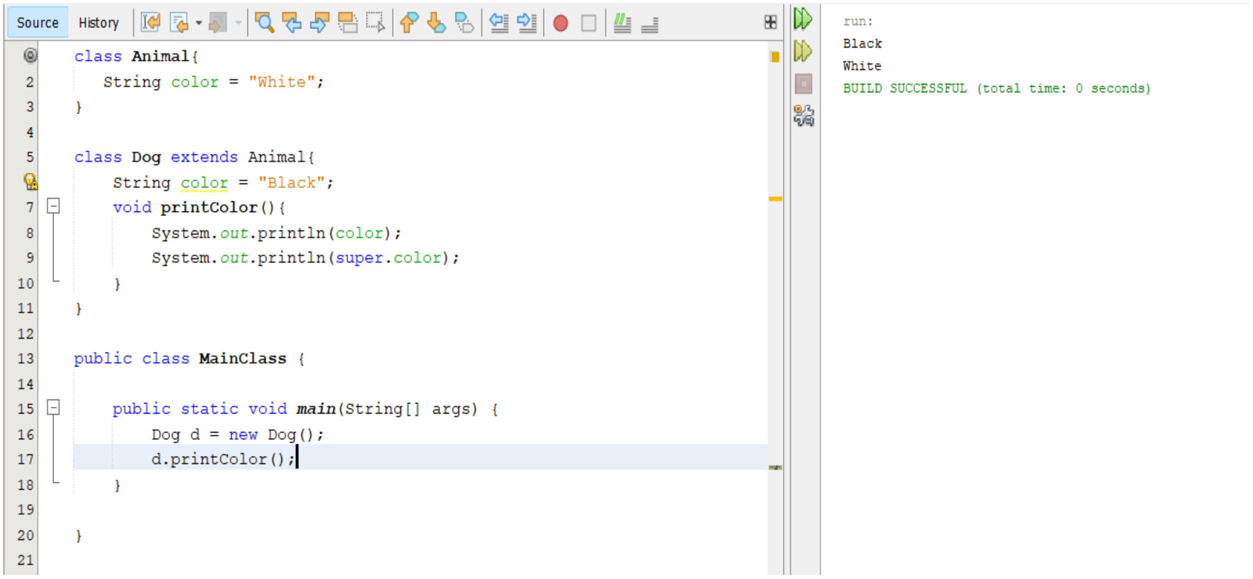
The super keyword refers to superclass (parent) objects.

It is used to call superclass methods, and to access the superclass constructor.

The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name.

Usage of Java super Keyword

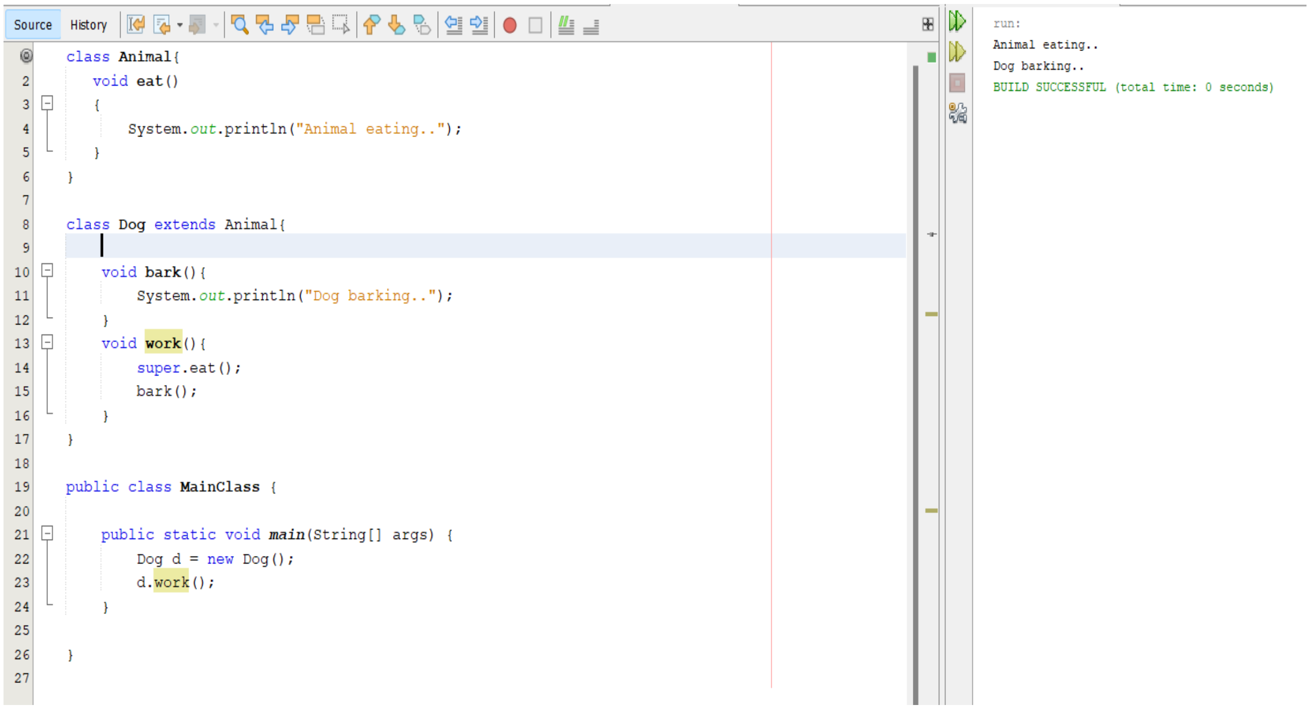
- super can be used to refer immediate parent class instance variable.



```
class Animal{
    String color = "White";
}

class Dog extends Animal{
    String color = "Black";
    void printColor(){
        System.out.println(color);
        System.out.println(super.color);
    }
}

public class MainClass {

    public static void main(String[] args) {
        Dog d = new Dog();
        d.printColor();
    }

}
```

```
run:
Black
White
BUILD SUCCESSFUL (total time: 0 seconds)
```

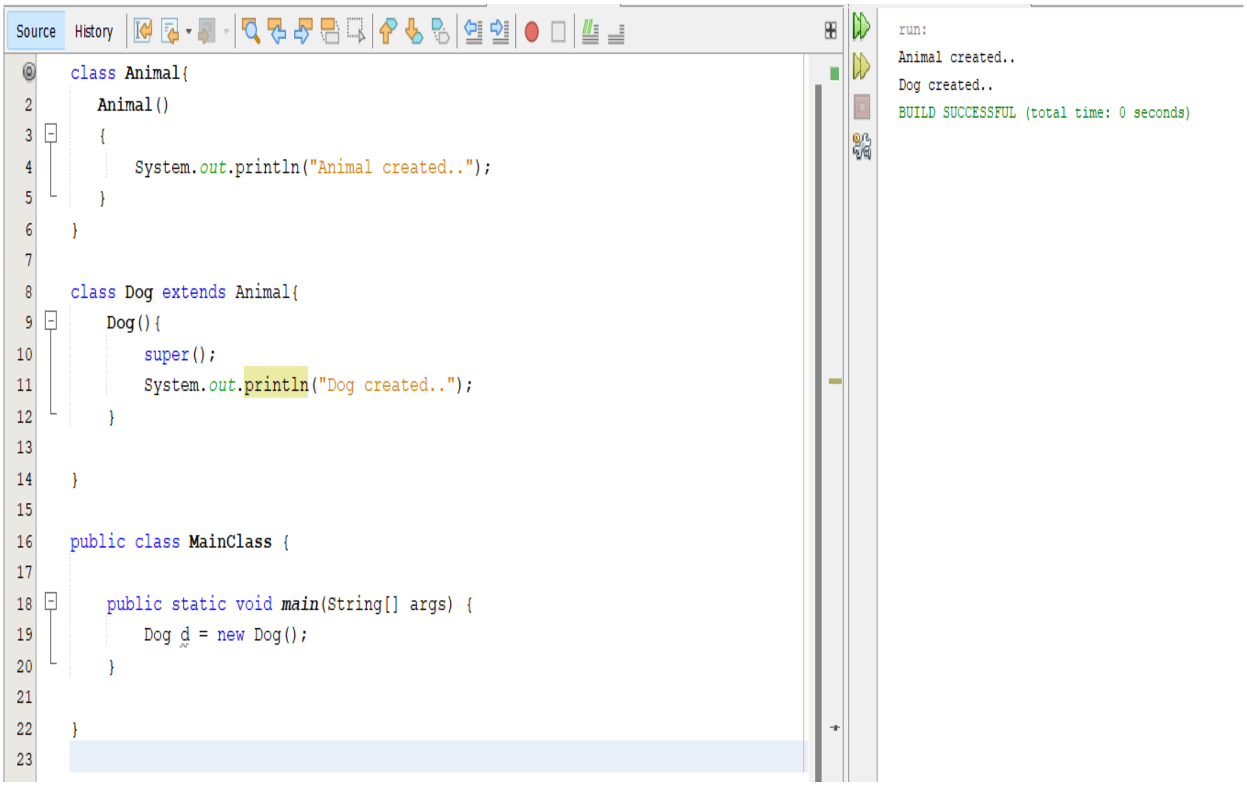
- super can be used to invoke immediate parent class method.



```
class Animal{
   void eat()
   {
       System.out.println("Animal eating..");
   }
}

class Dog extends Animal{

    void bark(){
        System.out.println("Dog barking..");
    }
    void work(){
        super.eat();
        bark();
    }
}

public class MainClass {

    public static void main(String[] args) {
        Dog d = new Dog();
        d.work();
    }

}
```

```
run:
Animal eating..
Dog barking..
BUILD SUCCESSFUL (total time: 0 seconds)
```

- super() can be used to invoke immediate parent class constructor



```java
class Animal{
   Animal()
   {
       System.out.println("Animal created..");
   }
}

class Dog extends Animal{
    Dog(){
        super();
        System.out.println("Dog created..");
    }

}

public class MainClass {

    public static void main(String[] args) {
        Dog d = new Dog();
    }

}
```

```
run:
Animal created..
Dog created..
BUILD SUCCESSFUL (total time: 0 seconds)
```

38.What is encapsulation?

Ans: *Encapsulation in Java* is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

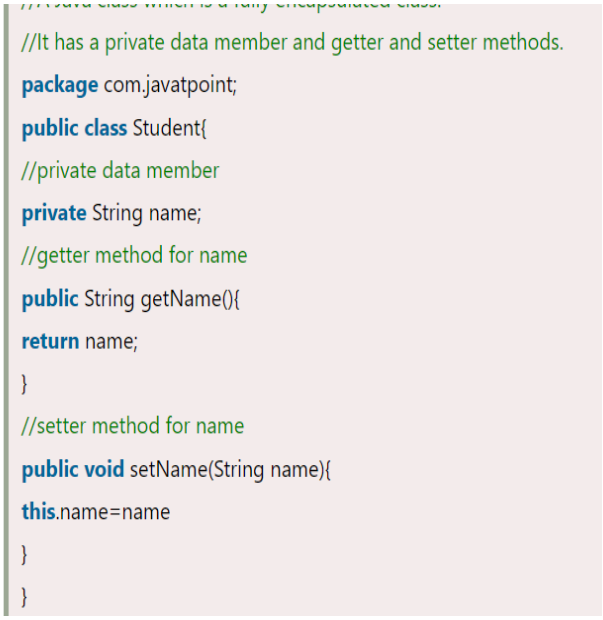
We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

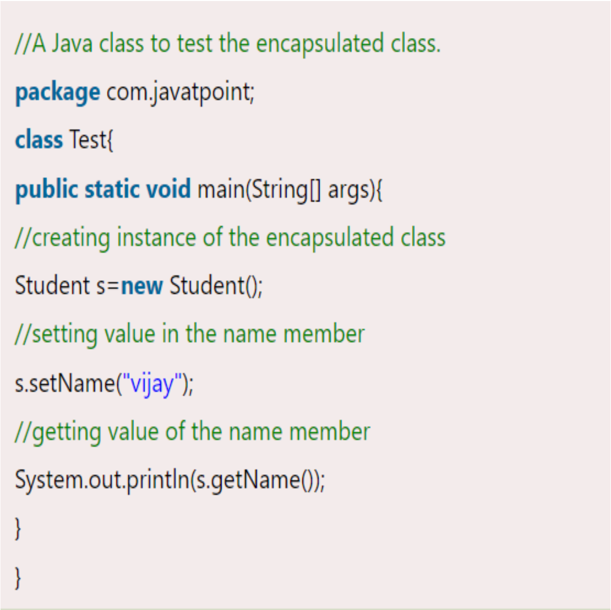
It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.



```
//A Java class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
package com.javatpoint;
public class Student{
//private data member
private String name;
//getter method for name
public String getName(){
return name;
}
//setter method for name
public void setName(String name){
this.name=name
}
}
```



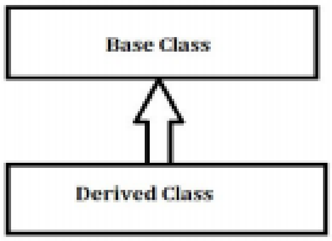
```
//A Java class to test the encapsulated class.
package com.javatpoint;
class Test{
public static void main(String[] args){
//creating instance of the encapsulated class
Student s=new Student();
//setting value in the name member
s.setName("vijay");
//getting value of the name member
System.out.println(s.getName());
}
}
```

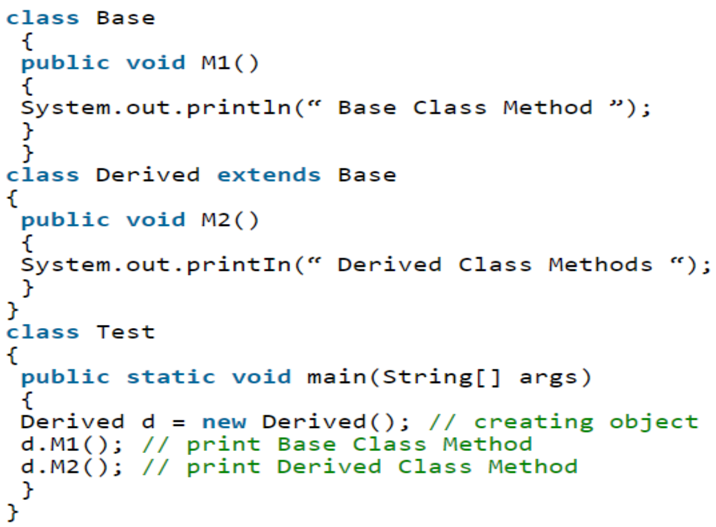
39.What is Inheritance in Java?

Ans: Inheritance in Java is a concept that acquires the properties from one class to other classes; for example, the relationship between father and son.

In Java, a class can inherit attributes and methods from another class. The class that inherits the properties is known as the sub-class or the child class. The class from which the properties are inherited is known as the superclass or the parent class.

In Inheritance, the properties of the base class are acquired by the derived classes.



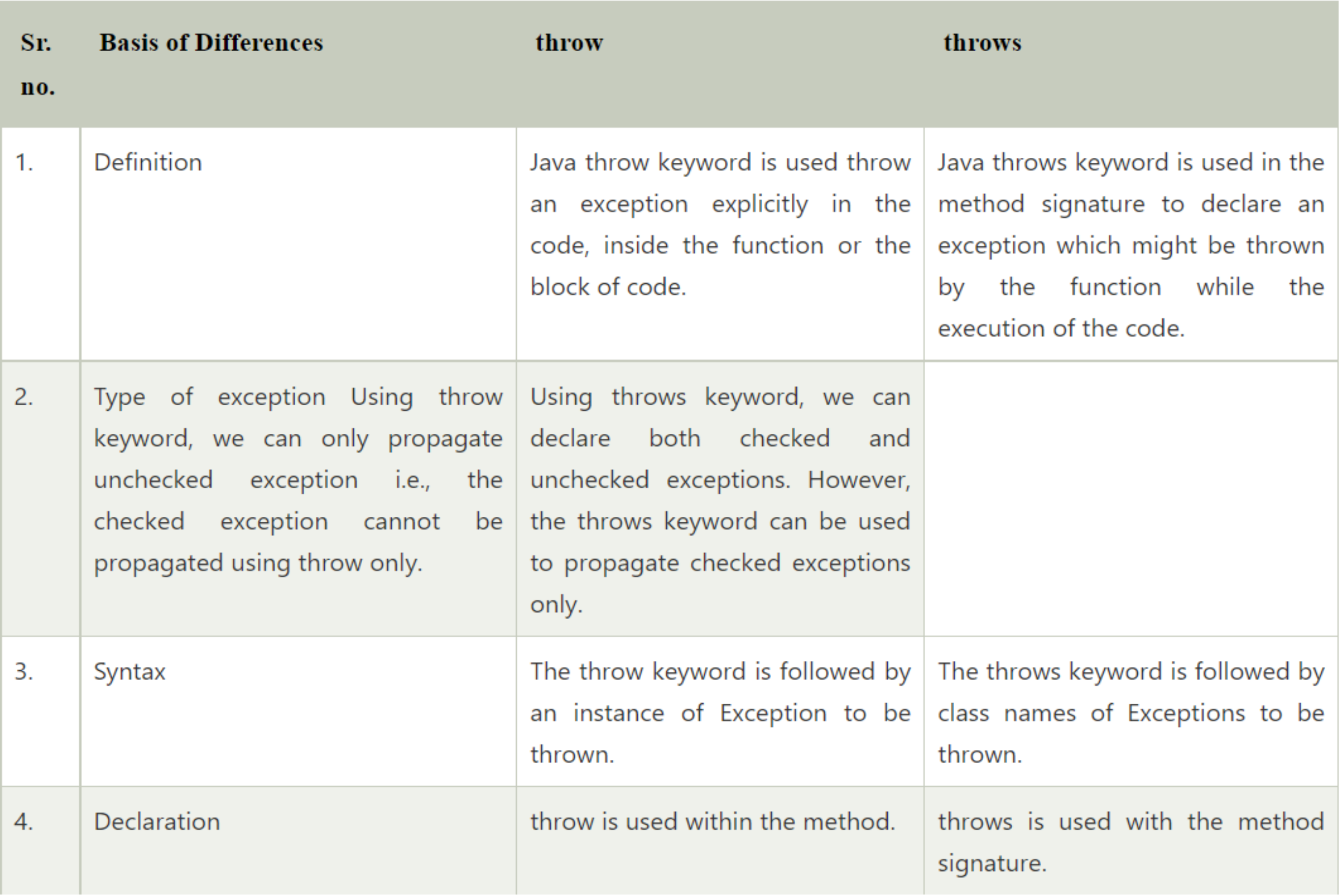


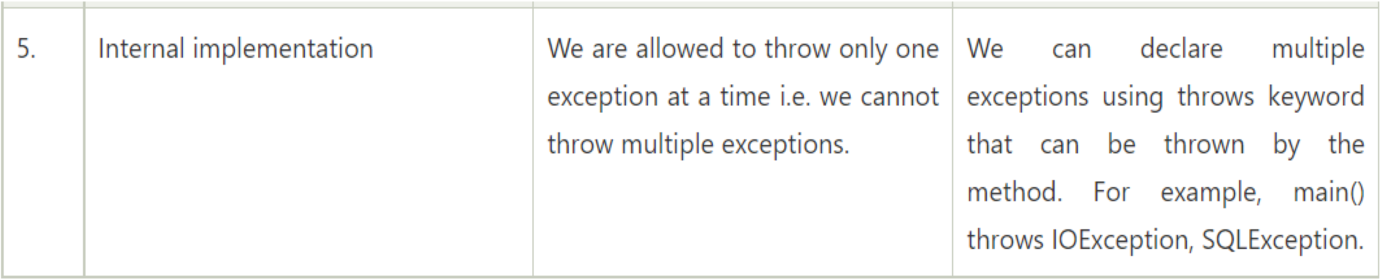
```
class Base
 {
 public void M1()
 {
 System.out.println(“ Base Class Method ”);
 }
 }
class Derived extends Base
{
 public void M2()
 {
 System.out.printIn(“ Derived Class Methods “);
 }
}
class Test
{
 public static void main(String[] args)
 {
 Derived d = new Derived(); // creating object
 d.M1(); // print Base Class Method
 d.M2(); // print Derived Class Method
 }
}
```

40.Difference Between Throw and Throws.

Ans:

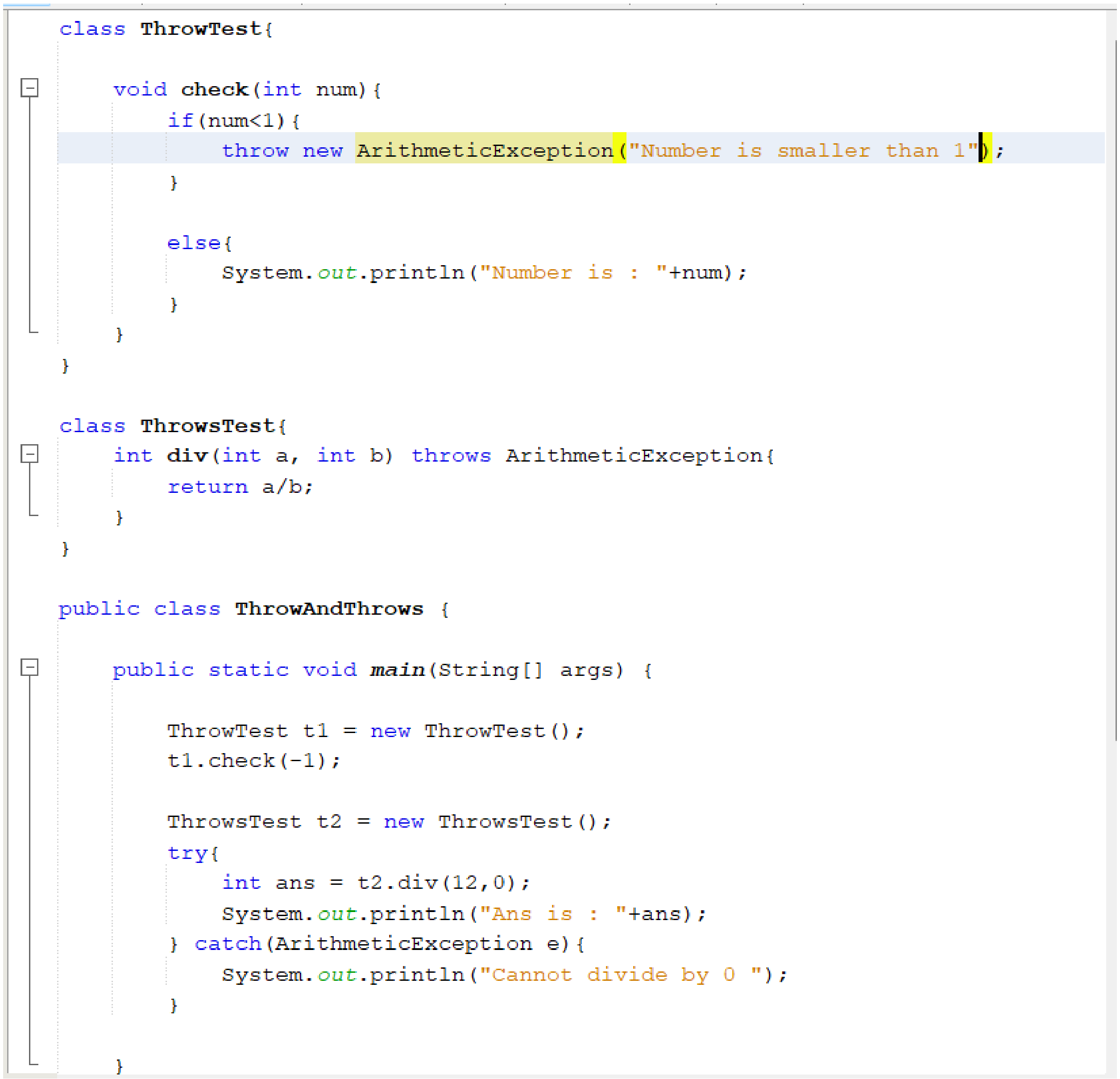
The throw and throws is the concept of exception handling where the throw keyword throw the exception explicitly from a method or a block of code whereas the throws keyword is used in signature of the method.





Sr. no.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

Example:



```
class ThrowTest{

    void check(int num){
        if(num<1){
            throw new ArithmeticException("Number is smaller than 1");
        }

        else{
            System.out.println("Number is : "+num);
        }
    }
}

class ThrowsTest{
    int div(int a, int b) throws ArithmeticException{
        return a/b;
    }
}

public class ThrowAndThrows {

    public static void main(String[] args) {

        ThrowTest t1 = new ThrowTest();
        t1.check(-1);

        ThrowsTest t2 = new ThrowsTest();
        try{
            int ans = t2.div(12,0);
            System.out.println("Ans is : "+ans);
        } catch(ArithmeticException e){
            System.out.println("Cannot divide by 0 ");
        }

    }
```

41.For java what does it mean by write once, run anywhere.

Ans: JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in Java code. JVM is a part of the JRE(Java Runtime Environment).

Java applications are called **WORA (Write Once Run Anywhere)**. This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

In traditional programming languages like C, C++ when programs were compiled, they used to be converted into the code understood by the particular underlying hardware, so If we try to run the same code at another machine with different hardware, which understands different code will cause an error, so you have to re-compile the code to be understood by the new hardware.

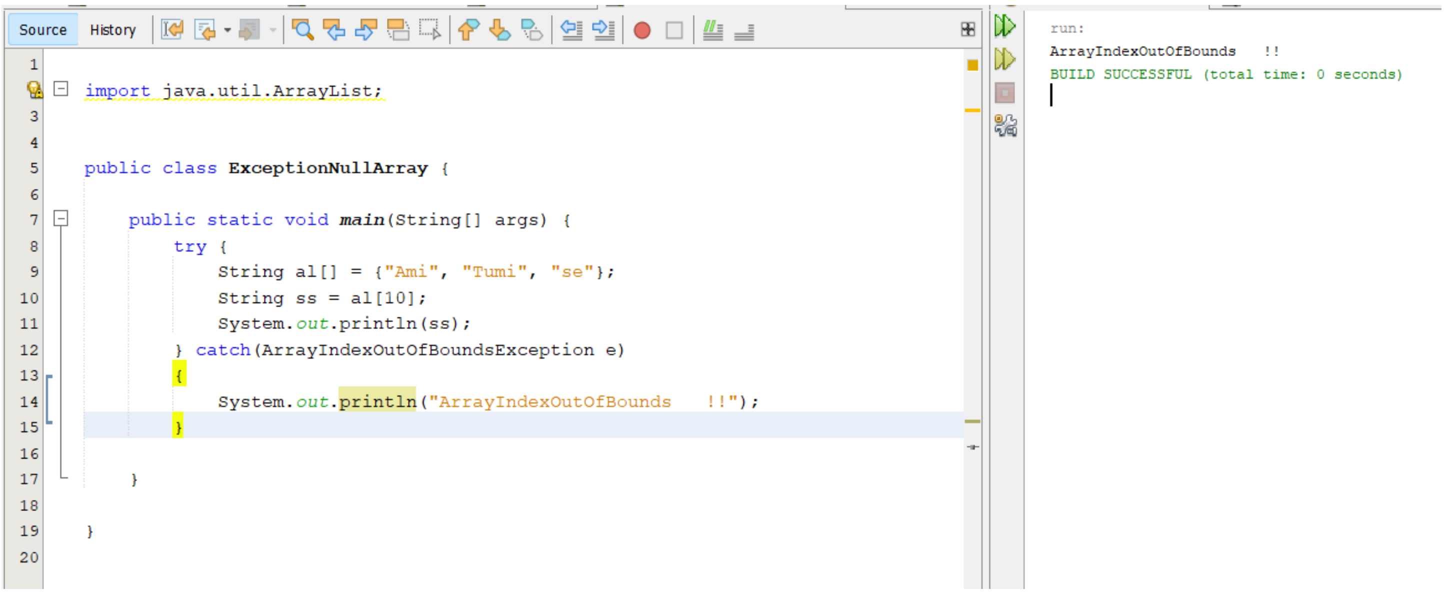
In Java, the program is not converted to code directly understood by Hardware, rather it is converted to bytecode(.class file) which is interpreted by JVM, so once compiled it generates bytecode file, which can be run anywhere (any machine) which has JVM(Java Virtual Machine) and hence it gets the nature of Write Once and Run Anywhere.

Related Question - 7

42.Write a program in Java that generates and catch "Null Pointer Exception" and "ArrayIndexOutofBounds " Exception.

Ans:

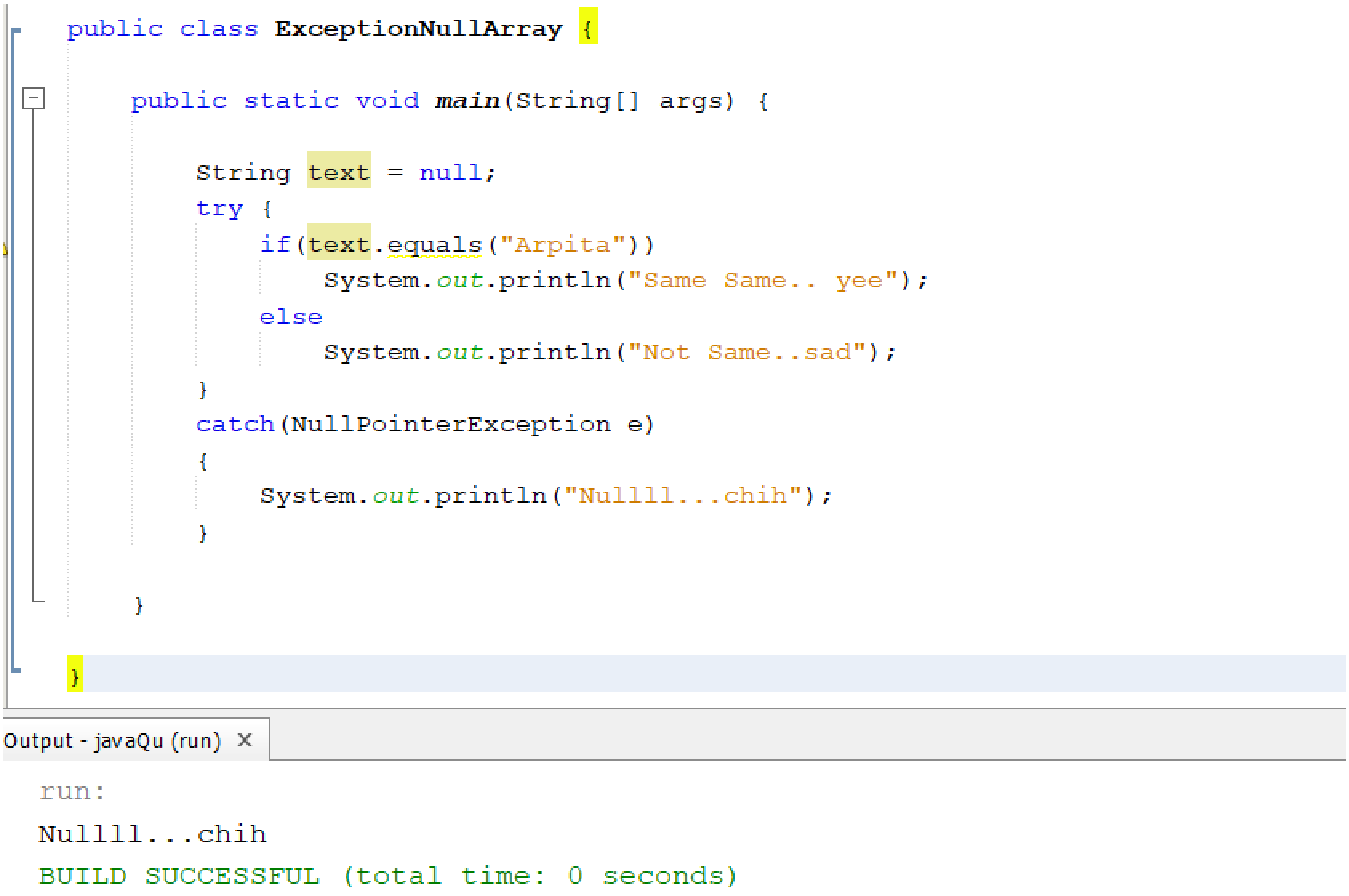
ArrayIndexOutOfBoundsException:



```
import java.util.ArrayList;


public class ExceptionNullArray {

    public static void main(String[] args) {
        try {
            String al[] = {"Ami", "Tumi", "se"};
            String ss = al[10];
            System.out.println(ss);
        } catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("ArrayIndexOutOfBounds   !!");
        }

    }

}
```

```
run:
ArrayIndexOutOfBounds   !!
BUILD SUCCESSFUL (total time: 0 seconds)
```

NullPointerException:



```
public class ExceptionNullArray {

    public static void main(String[] args) {

        String text = null;
        try {
            if(text.equals("Arpita"))
                System.out.println("Same Same.. yee");
            else
                System.out.println("Not Same..sad");
        }
        catch(NullPointerException e)
        {
            System.out.println("Nullll...chih");
        }

    }

}
```

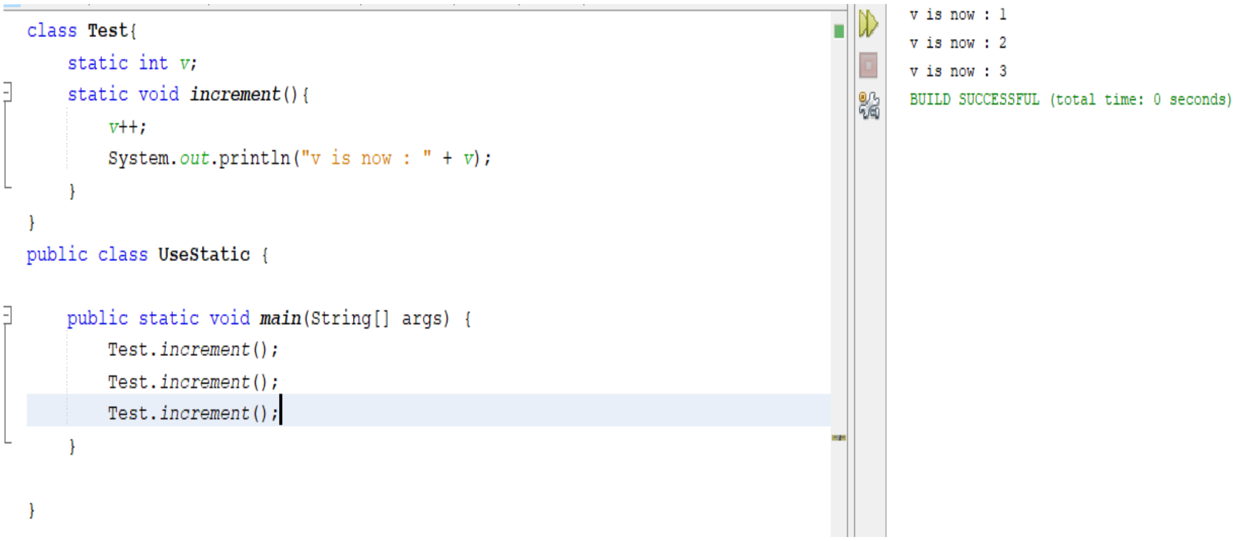
```
run:
Nullll...chih
BUILD SUCCESSFUL (total time: 0 seconds)
```

43.What is “Static”? Why would you use them? Provide example and explain.

Ans: Static Variables and Static Methods are class level and can be accessed through class name without need to create object of the class itself. In Java, it is possible to use the static keyword with methods, blocks, variables, as well as nested classes.

If any member in a class is declared as static, it means that even before the class is initiated, all the static members can be accessed and become active. In contrast to this, non-static members of the same class will cease to exist when there is no object or the object goes out of scope.

Why we use - The most important reason why static keywords are heavily used in Java is to efficiently manage memory. Generally, if you want to access variables or methods inside a class, you first need to create an instance or object of that class. However, there might be situations where you want to access only a couple of methods or variables of a class and you don’t want to create a new instance for that class just for accessing these members. This is where you can use the static keyword in Java.

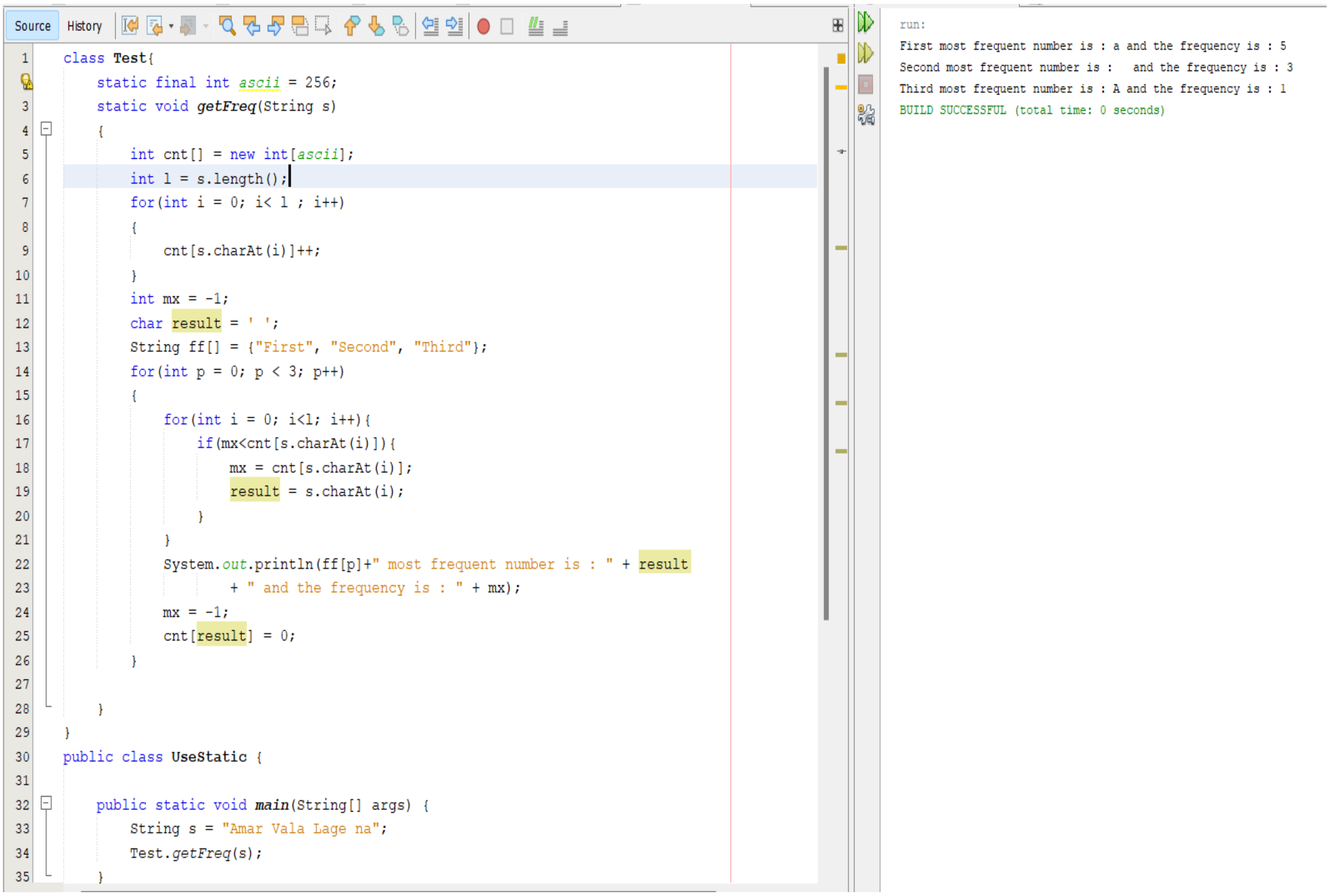


```
class Test{
    static int v;
    static void increment(){
        v++;
        System.out.println("v is now : " + v);
    }
}
public class UseStatic {

    public static void main(String[] args) {
        Test.increment();
        Test.increment();
        Test.increment();
    }


}
```

```
v is now : 1
v is now : 2
v is now : 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

44.Write a java code that will print the most three frequent characters in a given string.

Ans:



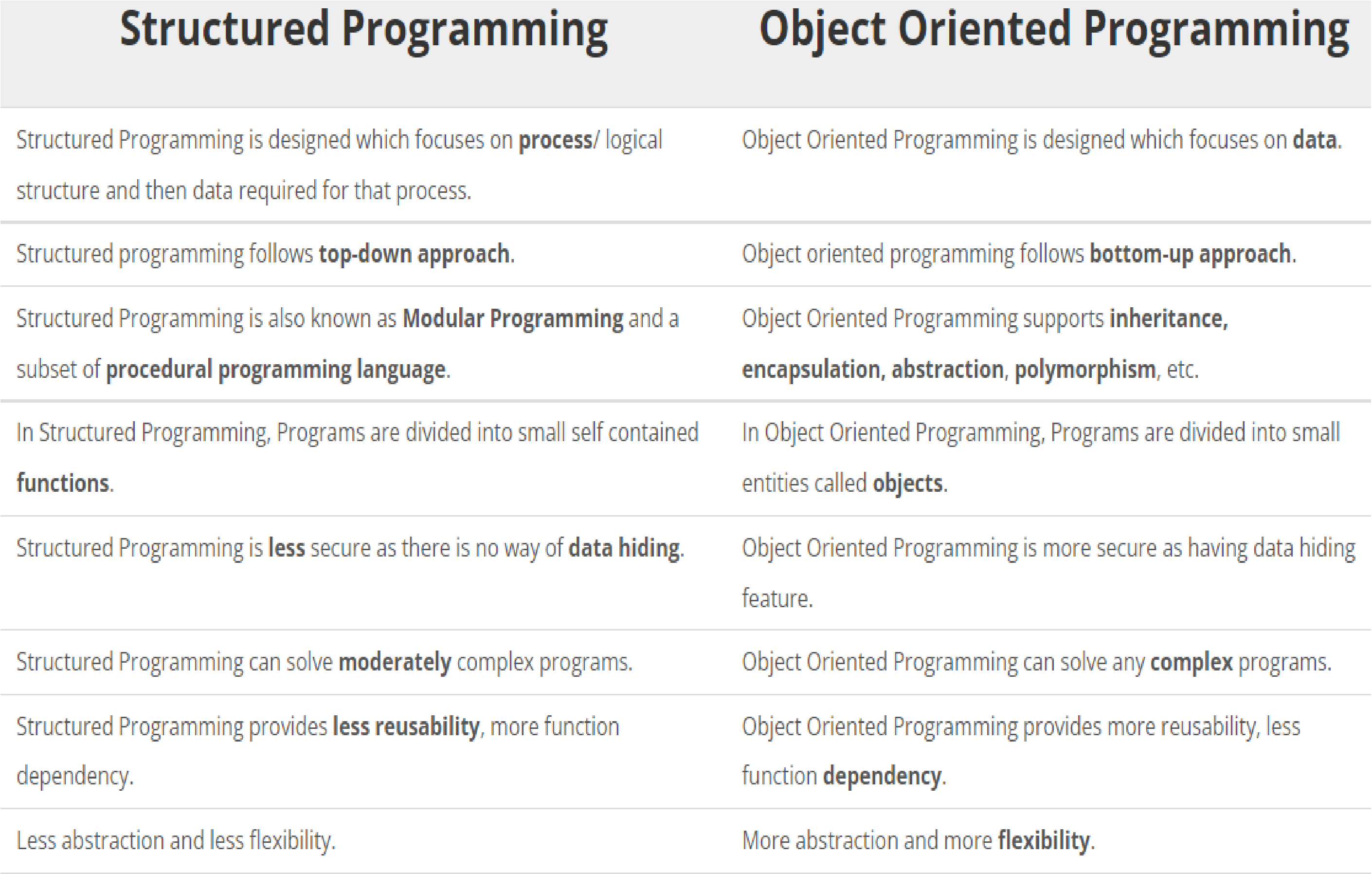
```
class Test{
    static final int ascii = 256;
    static void getFreq(String s)
    {
        int cnt[] = new int[ascii];
        int l = s.length();
        for(int i = 0; i< l ; i++)
        {
            cnt[s.charAt(i)]++;
        }
        int mx = -1;
        char result = ' ';
        String ff[] = {"First", "Second", "Third"};
        for(int p = 0; p < 3; p++)
        {
            for(int i = 0; i<l; i++){
                if(mx<cnt[s.charAt(i)]){
                    mx = cnt[s.charAt(i)];
                    result = s.charAt(i);
                }
            }
            System.out.println(ff[p]+" most frequent number is : " + result
                    + " and the frequency is : " + mx);
            mx = -1;
            cnt[result] = 0;
        }

    }
}
public class UseStatic {

    public static void main(String[] args) {
        String s = "Amar Vala Lage na";
        Test.getFreq(s);
    }
```

```
run:
First most frequent number is : a and the frequency is : 5
Second most frequent number is :   and the frequency is : 3
Third most frequent number is : A and the frequency is : 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

45.Why should you prefer OOP Language over structured programming language? Explain in your words.

Ans:

Object-Oriented Programming, as name suggests, is a different approach to programming that brings together data and functions that execute on them. It basically supports encapsulation, abstraction, inheritance, polymorphism, etc. It also includes data hiding feature therefore it is more secure. This model is based on real life entities that focuses on by whom task is to be done rather than focusing on what to do.



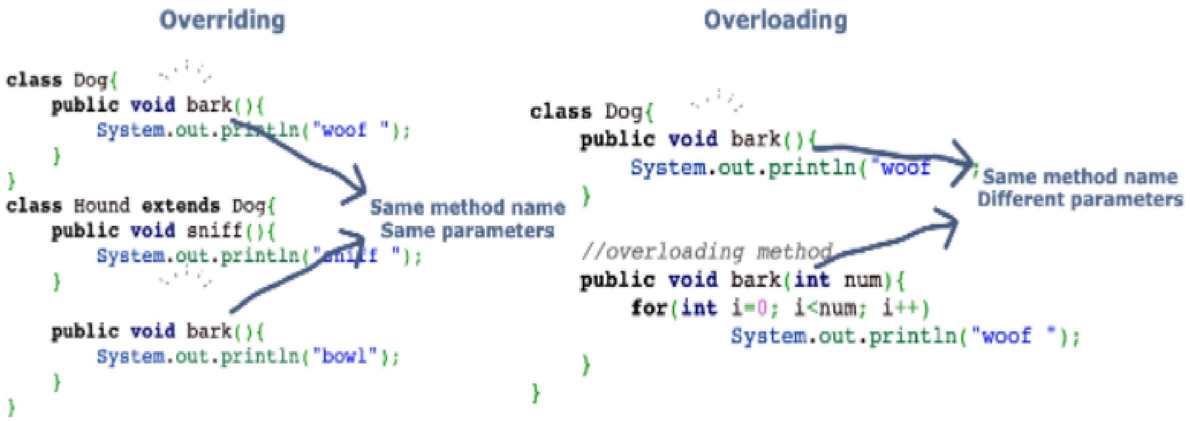
Structured Programming	Object Oriented Programming
Structured Programming is designed which focuses on **process**/ logical structure and then data required for that process.	Object Oriented Programming is designed which focuses on **data**.
Structured programming follows **top-down approach**.	Object oriented programming follows **bottom-up approach**.
Structured Programming is also known as **Modular Programming** and a subset of **procedural programming language**.	Object Oriented Programming supports **inheritance, encapsulation**, **abstraction**, **polymorphism**, etc.
In Structured Programming, Programs are divided into small self contained **functions**.	In Object Oriented Programming, Programs are divided into small entities called **objects**.
Structured Programming is **less** secure as there is no way of **data hiding**.	Object Oriented Programming is more secure as having data hiding feature.
Structured Programming can solve **moderately** complex programs.	Object Oriented Programming can solve any **complex** programs.
Structured Programming provides **less reusability**, more function dependency.	Object Oriented Programming provides more reusability, less function **dependency**.
Less abstraction and less flexibility.	More abstraction and more **flexibility**.

46.What is the difference between method overloading vs method overriding?

Ans:

Overloading occurs when two or more methods in one class have the same method name but different parameters.

Overriding occurs when two methods have the same method name and parameters. One of the methods is in the parent class, and the other is in the child class. Overriding allows a child class to provide the specific implementation of a method that is *already* present in its parent class.

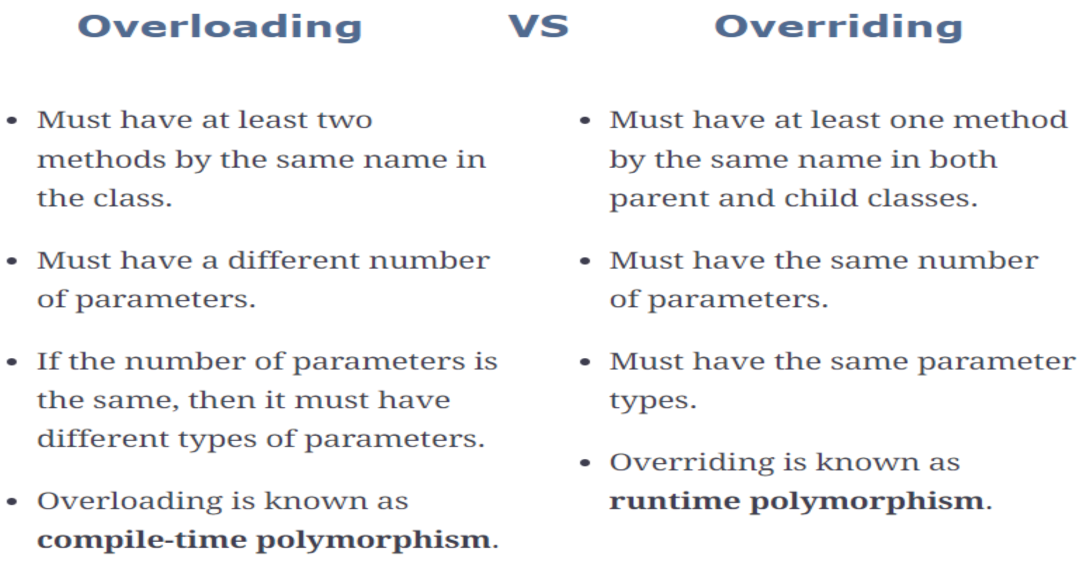


Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```



Overloading VS Overriding

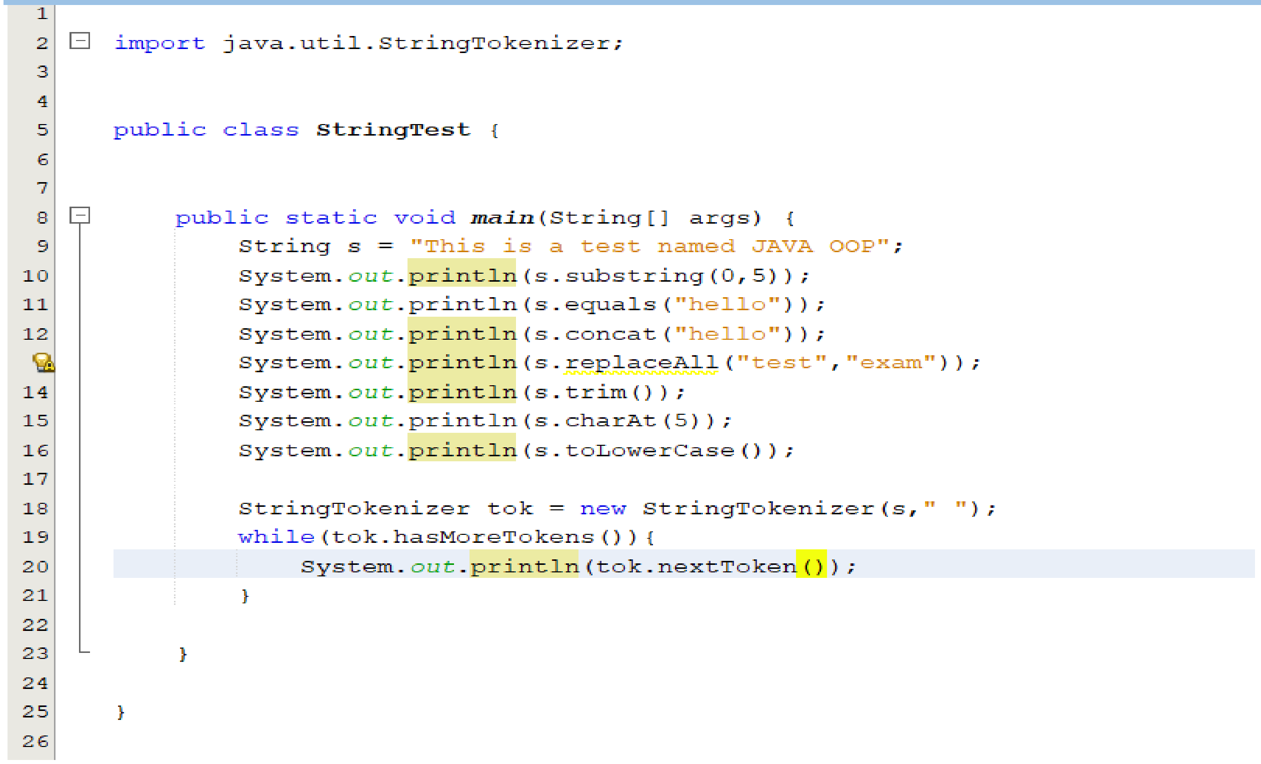
Overloading

- Must have at least two methods by the same name in the class.
- Must have a different number of parameters.
- If the number of parameters is the same, then it must have different types of parameters.
- Overloading is known as **compile-time polymorphism**.

Overriding

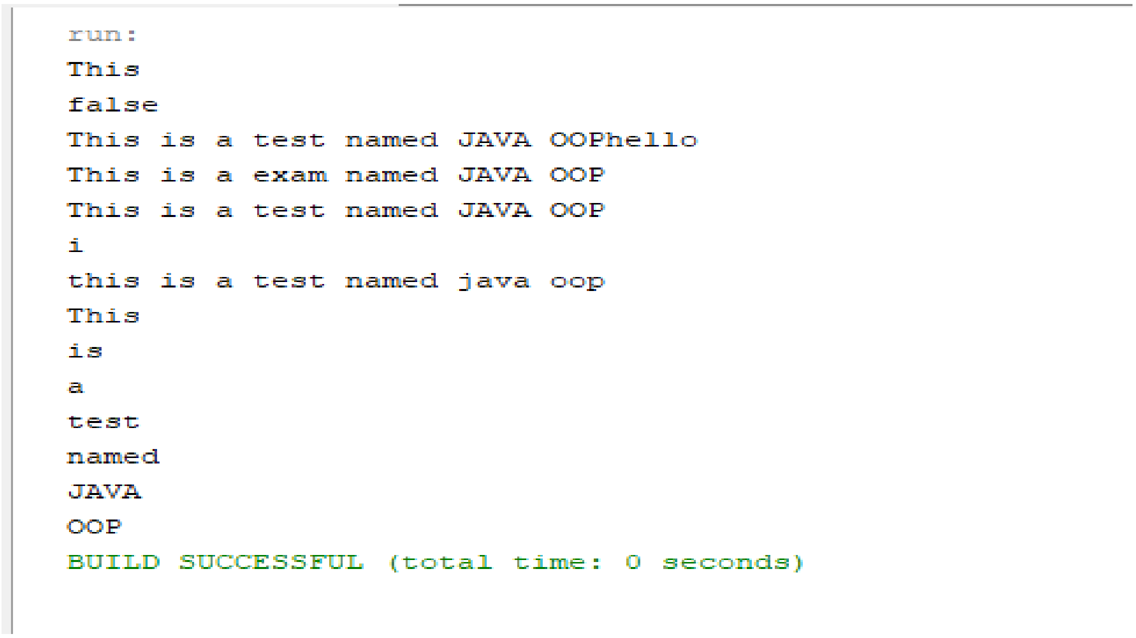
- Must have at least one method by the same name in both parent and child classes.
- Must have the same number of parameters.
- Must have the same parameter types.
- Overriding is known as **runtime polymorphism**.

47.Write Output of the following Java Program sequence.



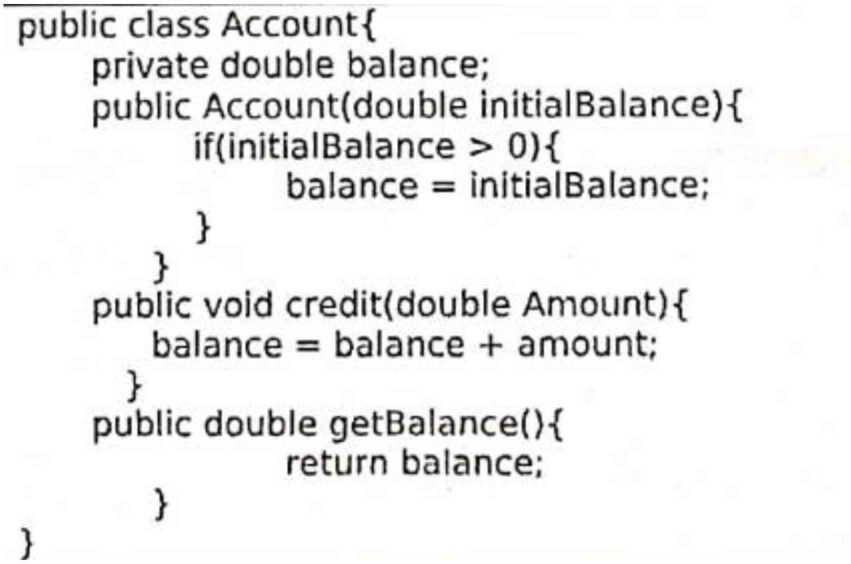
```java

import java.util.StringTokenizer;


public class StringTest {


    public static void main(String[] args) {
        String s = "This is a test named JAVA OOP";
        System.out.println(s.substring(0,5));
        System.out.println(s.equals("hello"));
        System.out.println(s.concat("hello"));
        System.out.println(s.replaceAll("test","exam"));
        System.out.println(s.trim());
        System.out.println(s.charAt(5));
        System.out.println(s.toLowerCase());

        StringTokenizer tok = new StringTokenizer(s," ");
        while(tok.hasMoreTokens()){
            System.out.println(tok.nextToken());
        }

    }

}

```

Ans:



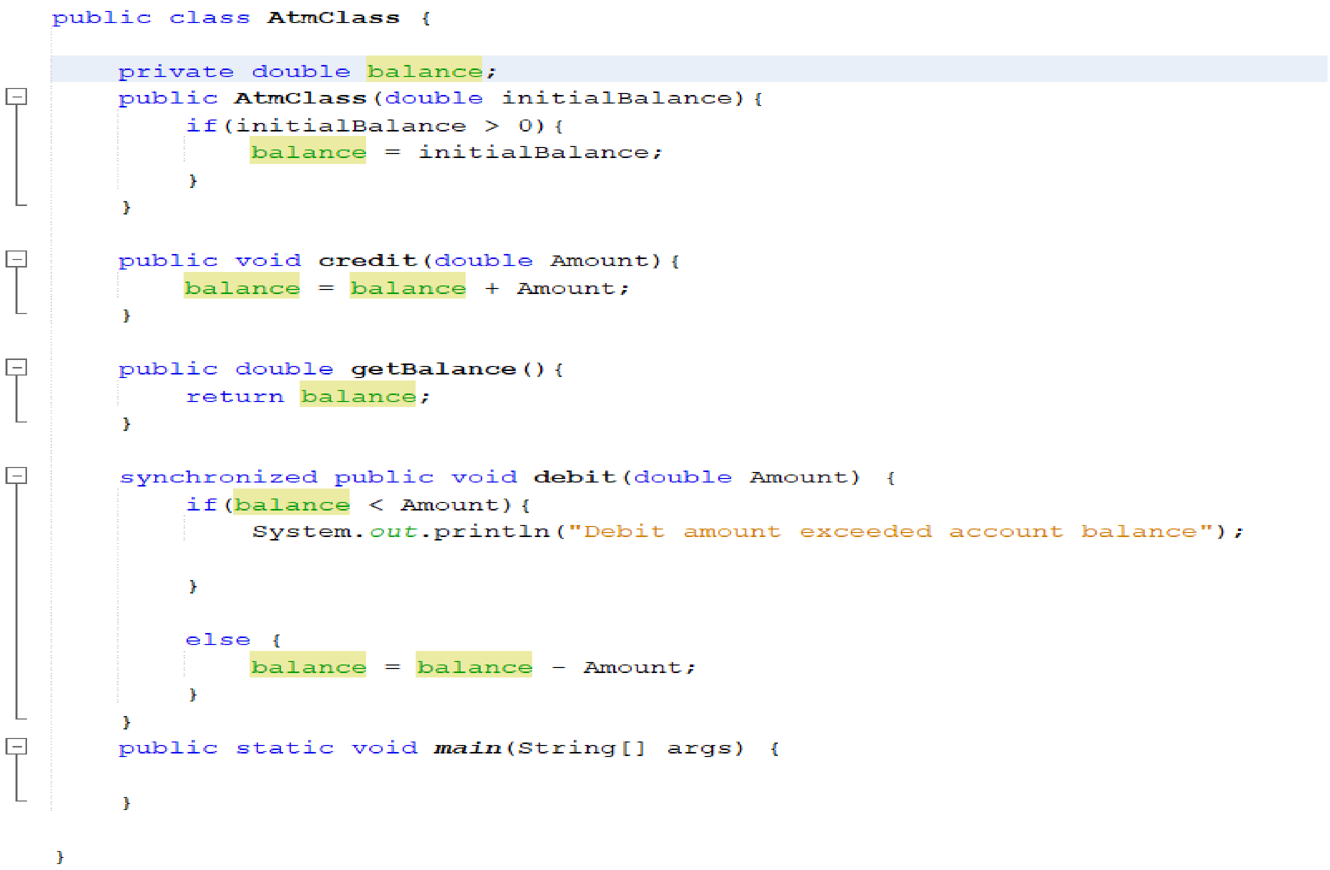
```
run:
This
false
This is a test named JAVA OOPhello
This is a exam named JAVA OOP
This is a test named JAVA OOP
i
this is a test named java oop
This
is
a
test
named
JAVA
OOP
BUILD SUCCESSFUL (total time: 0 seconds)
```

48. Modify the class Account to provide a method called debit that withdraws money from an Account. Ensure that the debit amount does not exceed the account's balance, if it does, the balance should be left unchanged and the method should print a message indicating "Debit amount exceeded account balance".



```
public class Account{
      private double balance;
      public Account(double initialBalance){
            if(initialBalance > 0){
                  balance = initialBalance;
            }
         }
      public void credit(double Amount){
         balance = balance + amount;
       }
      public double getBalance(){
                  return balance;
         }
}
```

Ans:



```
public class AtmClass {

     private double balance;
     public AtmClass(double initialBalance){
          if(initialBalance > 0){
               balance = initialBalance;
          }
     }

     public void credit(double Amount){
          balance = balance + Amount;
     }

     public double getBalance(){
          return balance;
     }

     synchronized public void debit(double Amount)  {
          if(balance < Amount){
               System.out.println("Debit amount exceeded account balance");

          }

          else  {
               balance = balance - Amount;
          }
     }
     public static void main(String[] args)  {

     }

}
```

49. Write a program in JAVA to calculate the area of geometric shapes by maintaining following conventions–

i)Create a generic parent object Area with common parameters and methods.

ii)Create child objects Triangle and Rectangle by inheriting Area and specifying separate functions for respective objects.

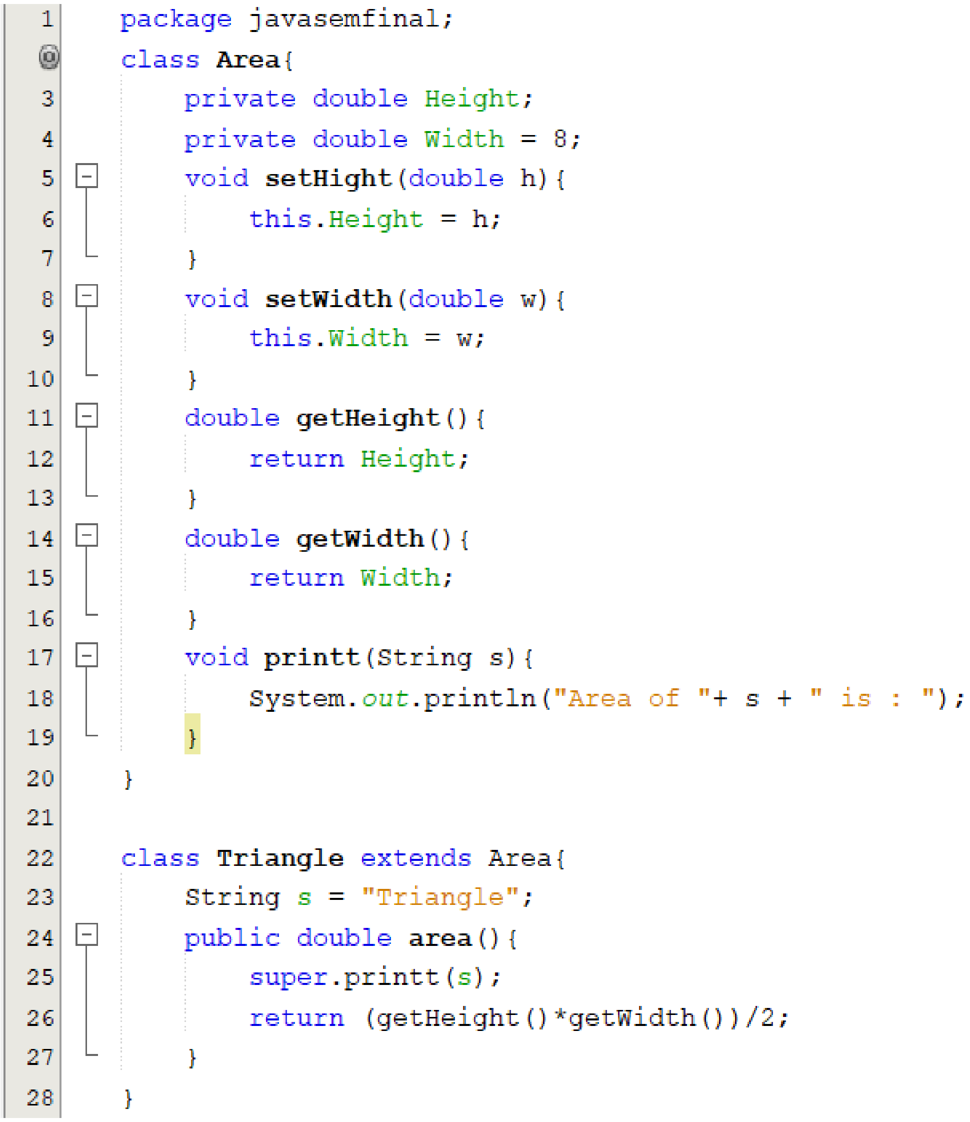
iii)Specify constructors for the object where possible.

iv)Create separate main class to call Triangle and Rectangle objects.

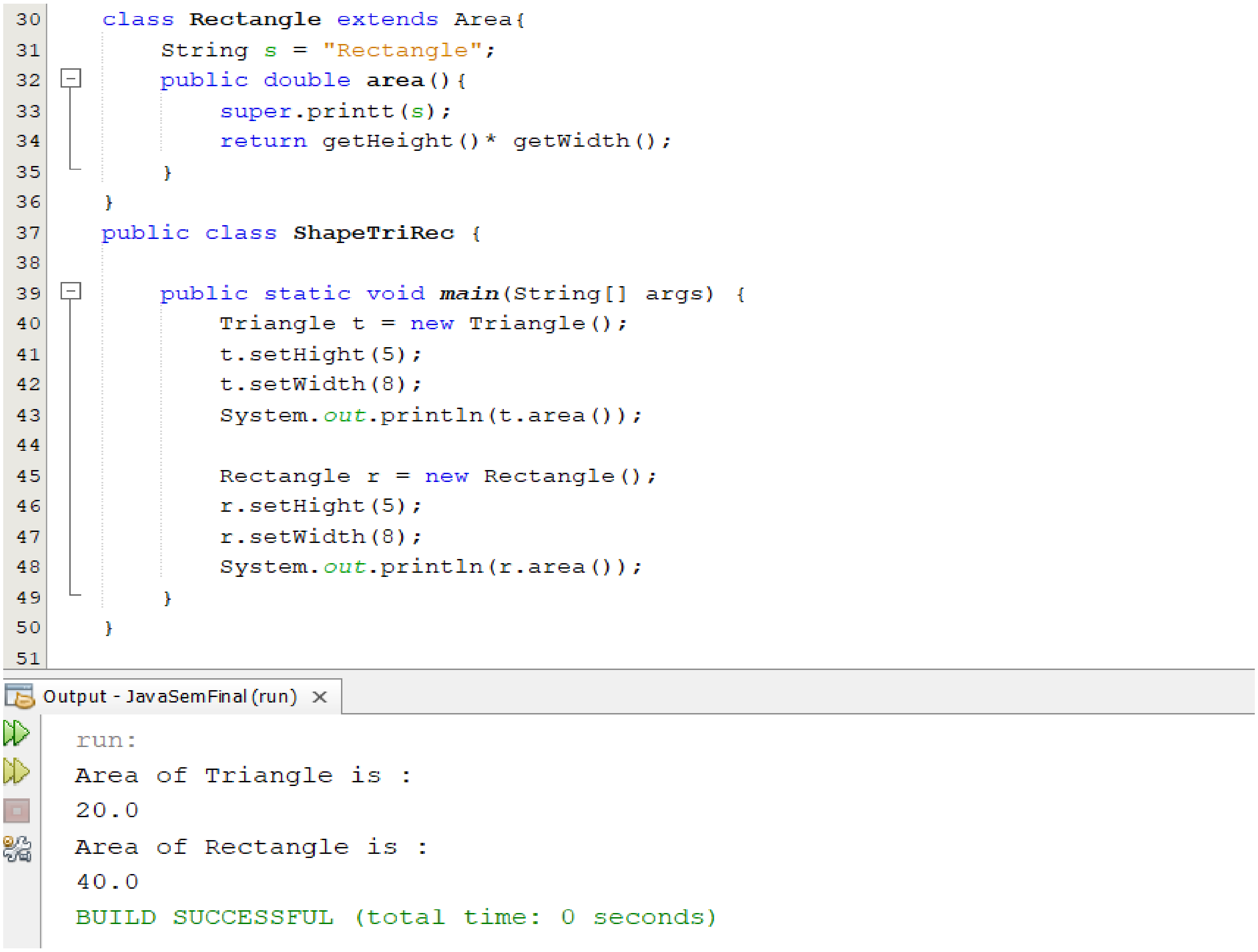
v)Show use of super and this where possible.

vi)Use your creativity.

Ans:



```java
package javasemfinal;
class Area{
    private double Height;
    private double Width = 8;
    void setHight(double h){
        this.Height = h;
    }
    void setWidth(double w){
        this.Width = w;
    }
    double getHeight(){
        return Height;
    }
    double getWidth(){
        return Width;
    }
    void printt(String s){
        System.out.println("Area of "+ s + " is : ");
    }
}

class Triangle extends Area{
    String s = "Triangle";
    public double area(){
        super.printt(s);
        return (getHeight()*getWidth())/2;
    }
}
```



```
class Rectangle extends Area{
    String s = "Rectangle";
    public double area(){
        super.printt(s);
        return getHeight()* getWidth();
    }
}
public class ShapeTriRec {

    public static void main(String[] args) {
        Triangle t = new Triangle();
        t.setHight(5);
        t.setWidth(8);
        System.out.println(t.area());

        Rectangle r = new Rectangle();
        r.setHight(5);
        r.setWidth(8);
        System.out.println(r.area());
    }
}
```

```
run:
Area of Triangle is :
20.0
Area of Rectangle is :
40.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

50. Show an example of Mutex lock using JAVA.

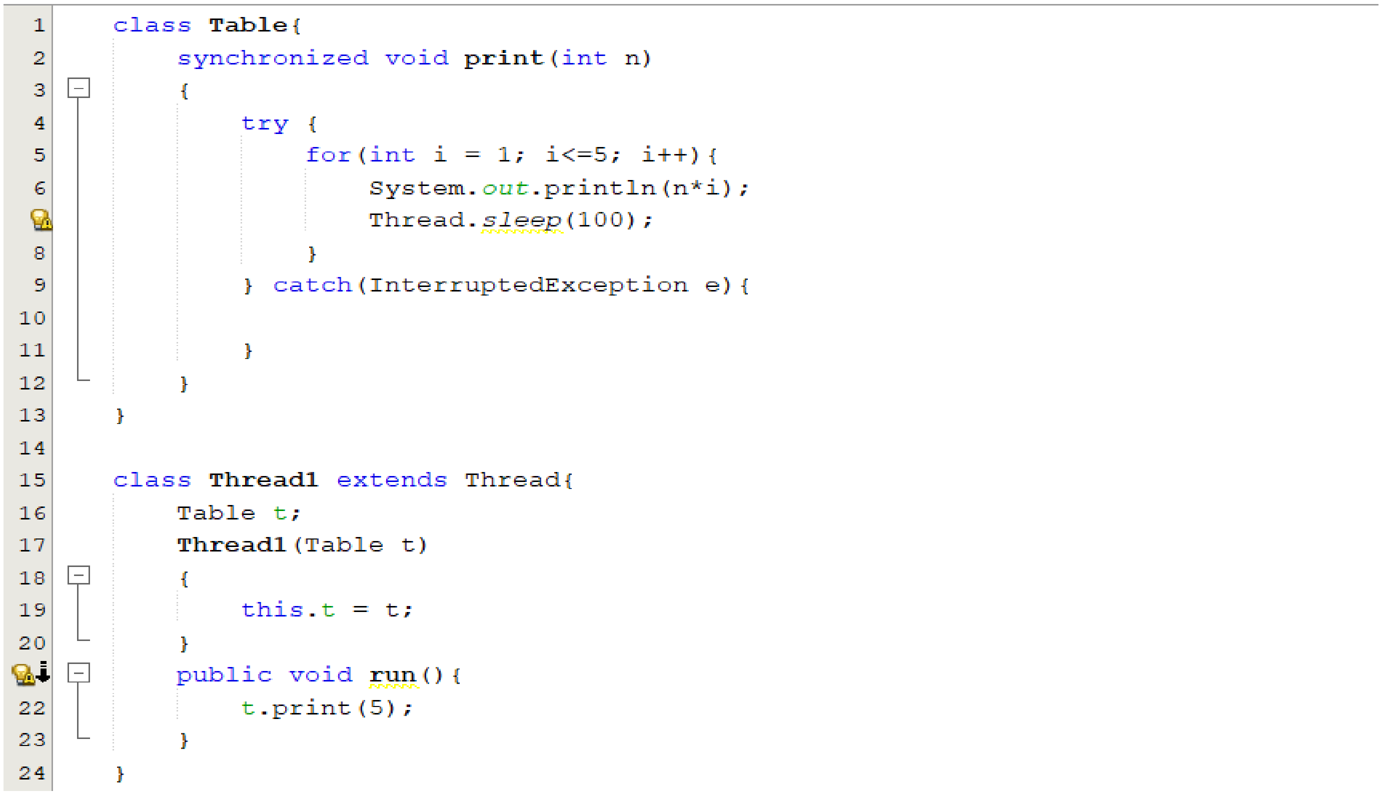
Ans:

In a multithreaded application, two or more threads may need to access a shared resource at the same time, resulting in unexpected behavior. A mutex (or mutual exclusion) is the simplest type of *synchronizer* – it **ensures that only one thread can execute the critical section of a computer program at a time**.

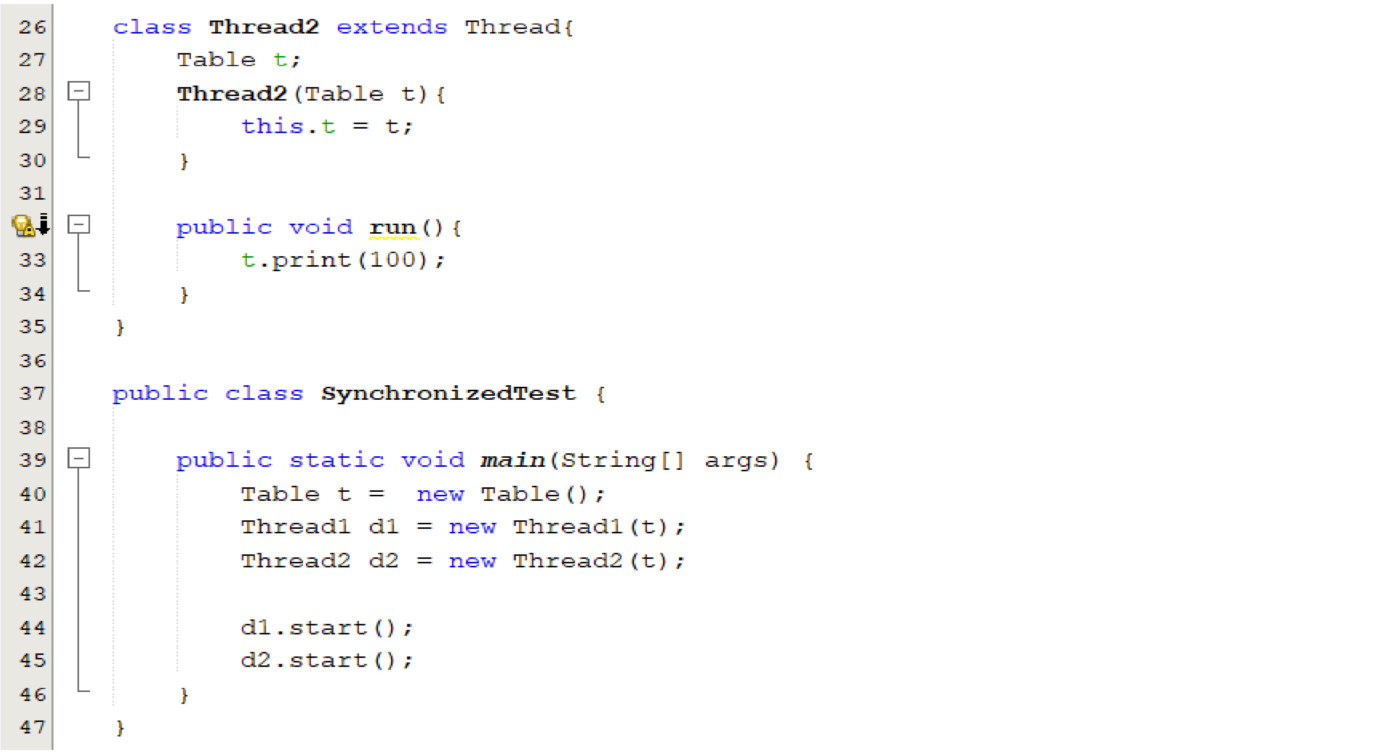
To access a critical section, a thread acquires the mutex, then accesses the critical section, and finally releases the mutex. In the meantime, **all other threads block till the mutex releases.** As soon as a thread exits the critical section, another thread can enter the critical section.

The simplest way to implement a mutex in Java is using synchronized keyword. Every object in Java has an intrinsic lock associated with it. **The *synchronized* method and the *synchronized* block use this intrinsic lock** to restrict the access of the critical section to only one thread at a time. Therefore, when a thread invokes a *synchronized* method or enters a *synchronized* block, it automatically acquires the lock. The lock releases when the method or block completes or an exception is thrown from them.

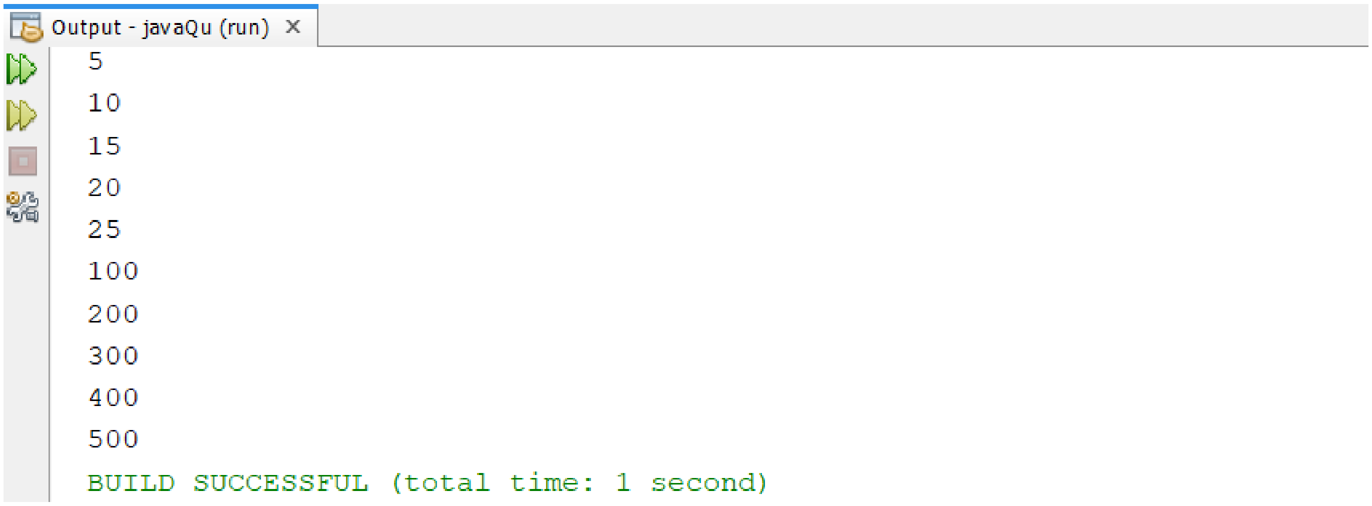
Example:



```java
class Table{
    synchronized void print(int n)
    {
        try {
            for(int i = 1; i<=5; i++){
                System.out.println(n*i);
                Thread.sleep(100);
            }
        } catch(InterruptedException e){

        }
    }
}

class Thread1 extends Thread{
    Table t;
    Thread1(Table t)
    {
        this.t = t;
    }
    public void run(){
        t.print(5);
    }
}
```



```java
class Thread2 extends Thread{
    Table t;
    Thread2(Table t){
        this.t = t;
    }

    public void run(){
        t.print(100);
    }
}

public class SynchronizedTest {

    public static void main(String[] args) {
        Table t =  new Table();
        Thread1 d1 = new Thread1(t);
        Thread2 d2 = new Thread2(t);

        d1.start();
        d2.start();
    }
}
```

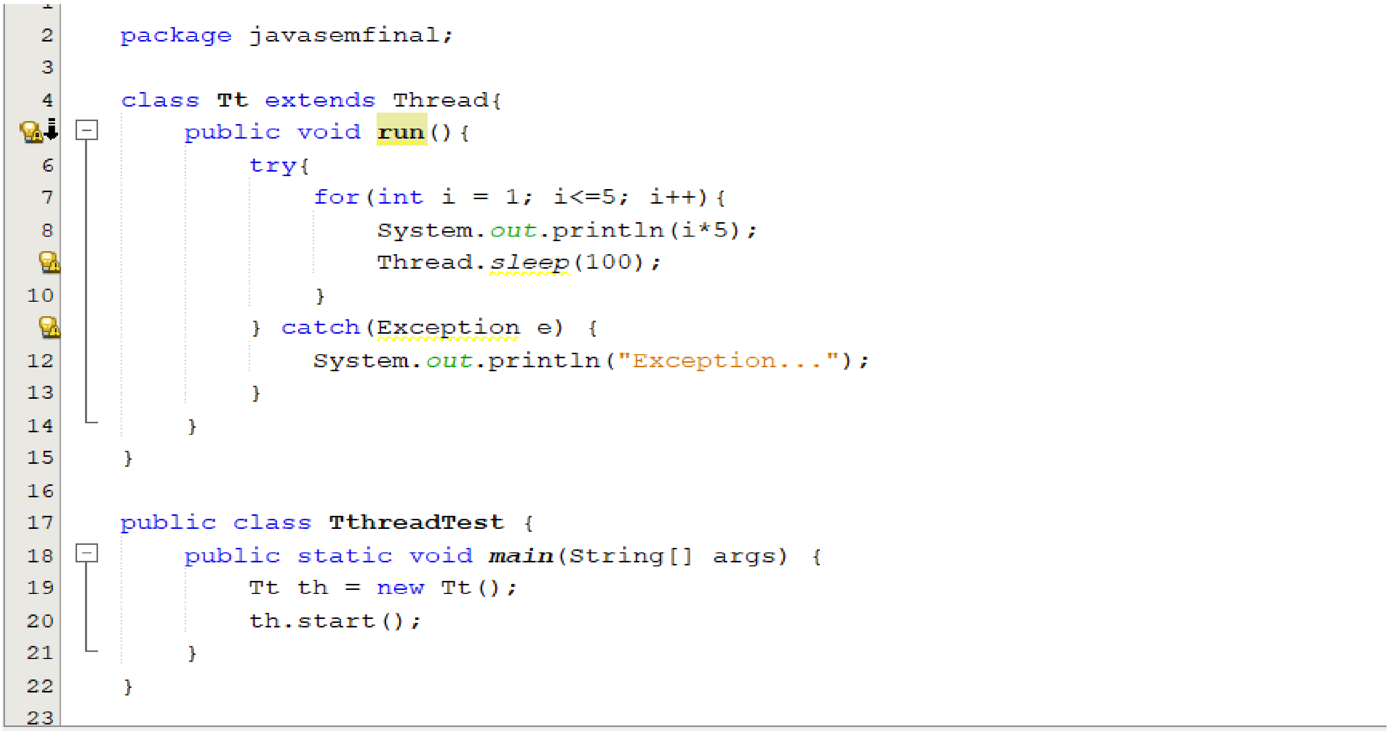


```
5
10
15
20
25
100
200
300
400
500
BUILD SUCCESSFUL (total time: 1 second)
```

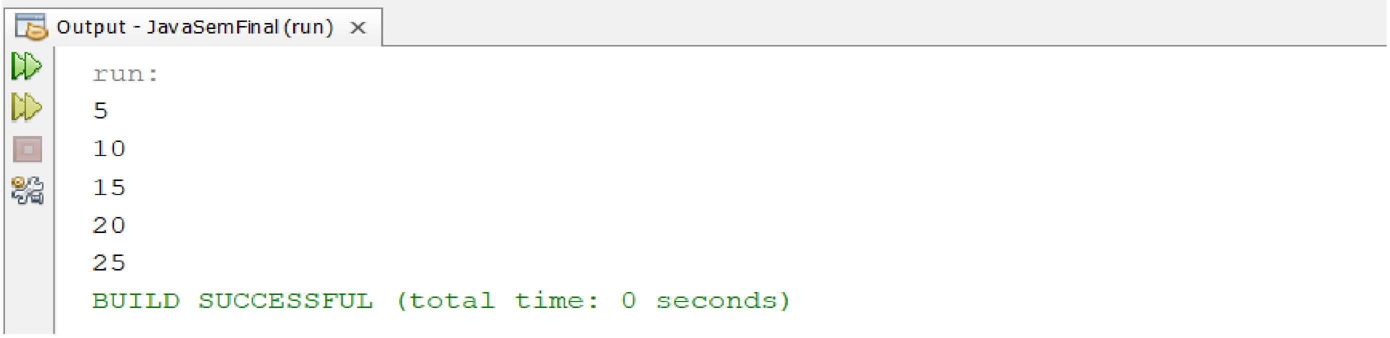
51. How many ways we can create threads? Show example.

Ans: There are two ways to create a thread:

1.By extending Thread class

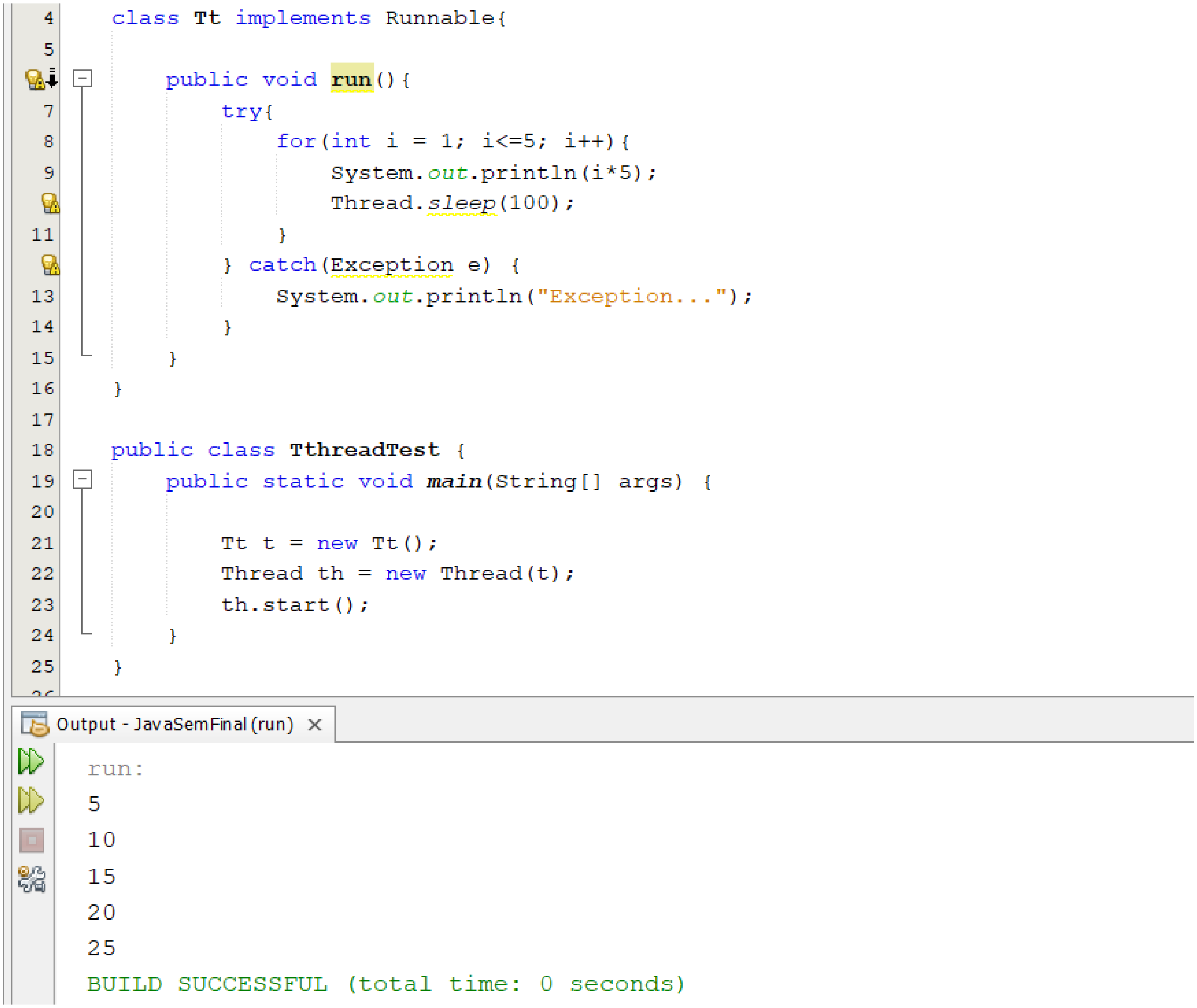


```java
package javasemfinal;

class Tt extends Thread{
    public void run(){
        try{
            for(int i = 1; i<=5; i++){
                System.out.println(i*5);
                Thread.sleep(100);
            }
        } catch(Exception e)  {
            System.out.println("Exception...");
        }
    }
}

public class TthreadTest {
    public static void main(String[] args)  {
        Tt th = new Tt();
        th.start();
    }
}
```



```
run:
5
10
15
20
25
BUILD SUCCESSFUL (total time: 0 seconds)
```

2.By implementing Runnable interface.

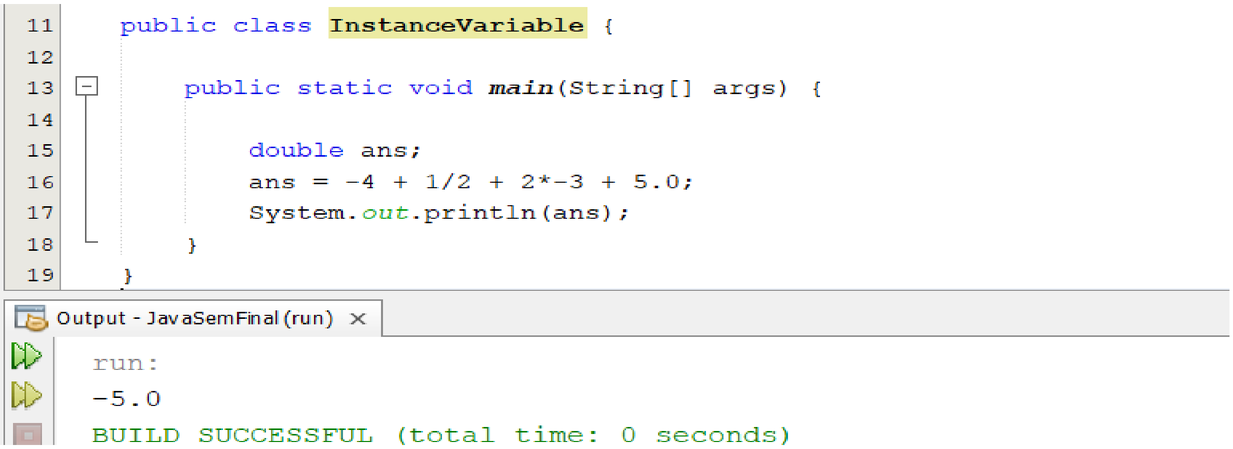


```
class Tt implements Runnable{

    public void run(){
        try{
            for(int i = 1; i<=5; i++){
                System.out.println(i*5);
                Thread.sleep(100);
            }
        } catch(Exception e) {
            System.out.println("Exception...");
        }
    }
}

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

        Tt t = new Tt();
        Thread th = new Thread(t);
        th.start();
    }
}
```

```
run:
5
10
15
20
25
BUILD SUCCESSFUL (total time: 0 seconds)
```

52. What is the type and value of the following expression? -4 + ½ + 2*-3 + 5.0.

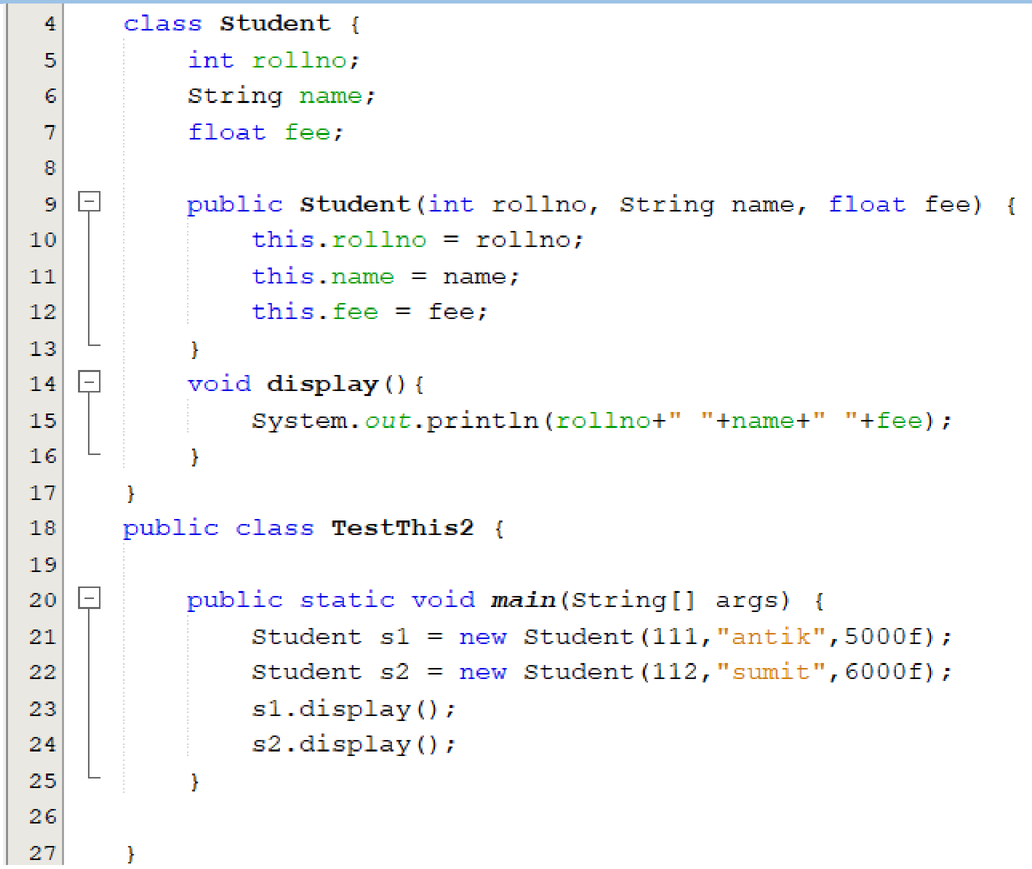
Ans: The type of the following expression is double and the value is -5.0.



```
public class InstanceVariable {

    public static void main(String[] args) {

        double ans;
        ans = -4 + 1/2 + 2*-3 + 5.0;
        System.out.println(ans);
    }
}
```

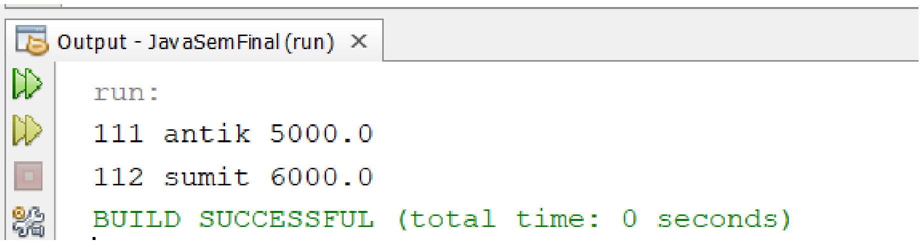
```
run:
-5.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

53. Write down the output of the following program.



```
class Student {
    int rollno;
    String name;
    float fee;

    public Student(int rollno, String name, float fee) {
        this.rollno = rollno;
        this.name = name;
        this.fee = fee;
    }
    void display(){
        System.out.println(rollno+" "+name+" "+fee);
    }
}
public class TestThis2 {

    public static void main(String[] args) {
        Student s1 = new Student(111,"antik",5000f);
        Student s2 = new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
    }

}
```

Ans:



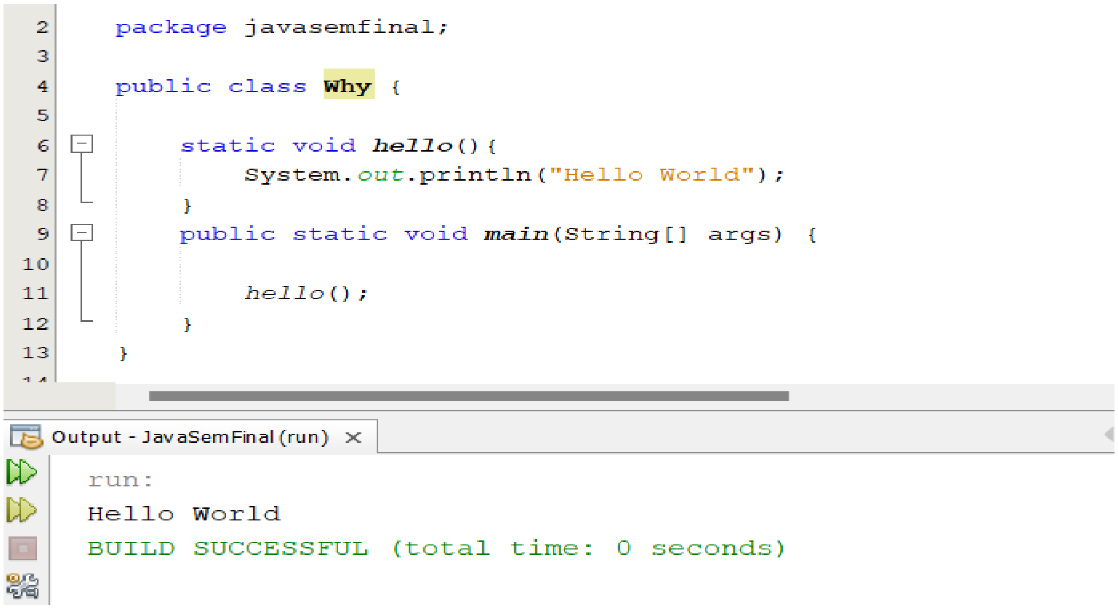
```
run:
111 antik 5000.0
112 sumit 6000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

54. Look at the code carefully. Does the code have any bug? If not, write no problem found. If yes, correct the code and explain why.

```
public class Why{
        void hello(){
                System.out.println(“Hello World”);
        }
        public static void main(String[] args) {
                hello();
        }
}
```

Ans: Here the problem is hello() method is a non-static method. Non-static method course can-not be referenced from a static context.

So, the solution is:



```java
package javasemfinal;

public class Why {

    static void hello(){
        System.out.println("Hello World");
    }
    public static void main(String[] args) {

        hello();
    }
}
```

```
run:
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

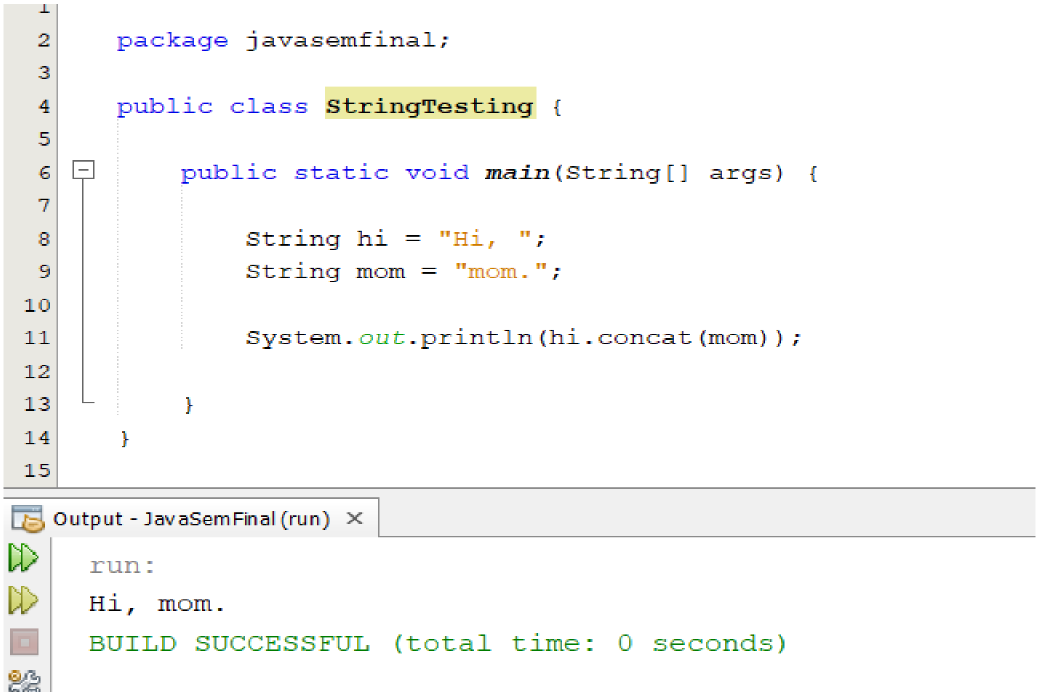
55. Show two ways to concatenate the following two strings together to get the string "Hi, mom.":

String hi = "Hi, ";

String mom = "mom.";

Ans:

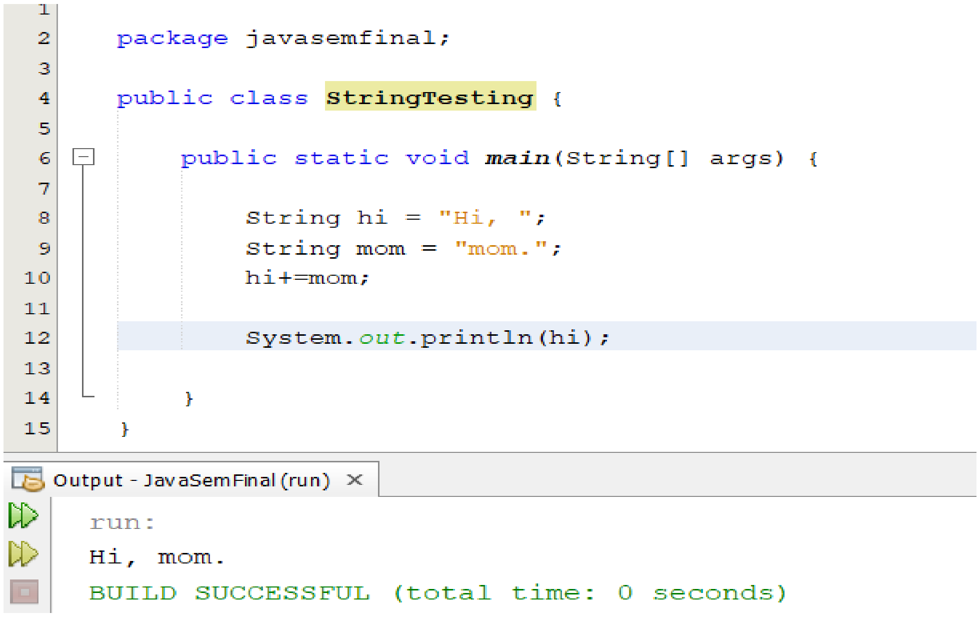
1st way:



```java
package javasemfinal;

public class StringTesting {

    public static void main(String[] args) {

        String hi = "Hi, ";
        String mom = "mom.";

        System.out.println(hi.concat(mom));

    }
}
```

```
run:
Hi, mom.
BUILD SUCCESSFUL (total time: 0 seconds)
```

2nd way:

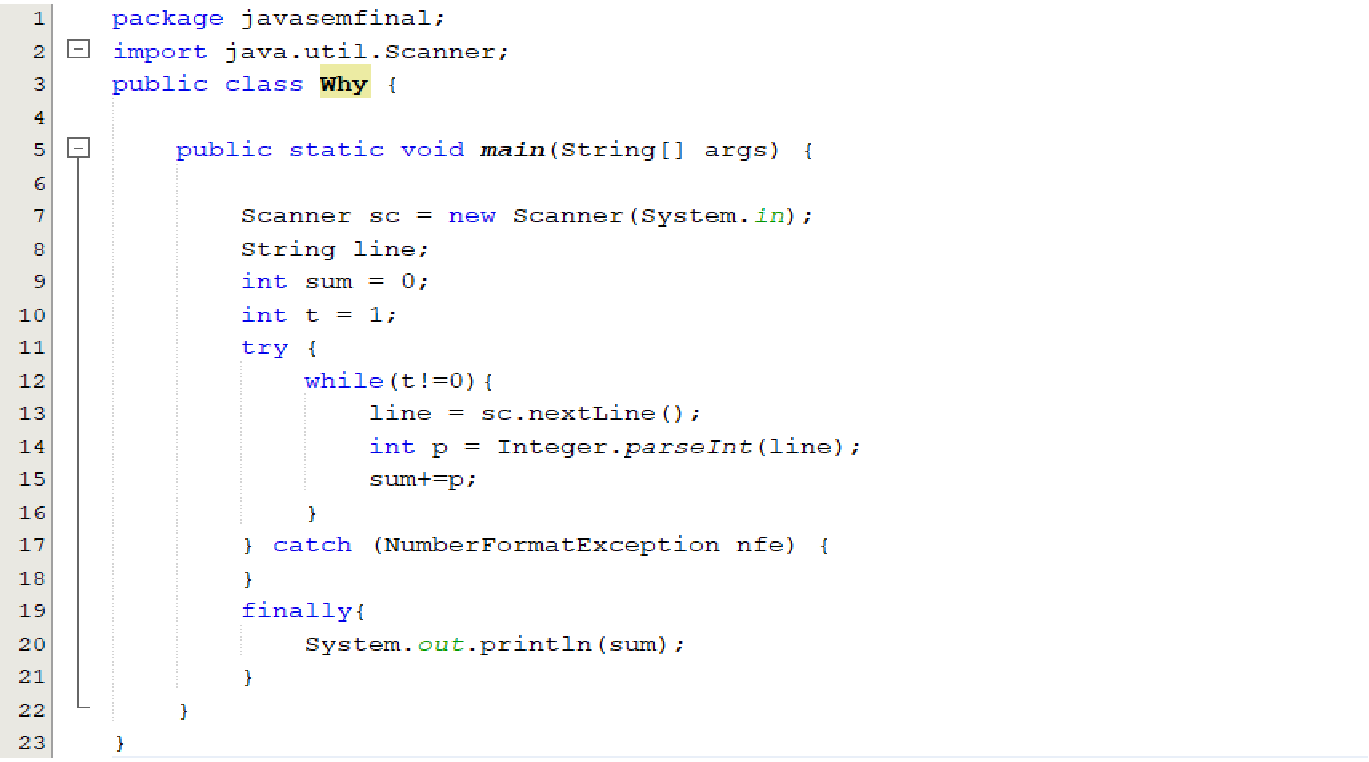


```

package javasemfinal;

public class StringTesting {

    public static void main(String[] args) {

        String hi = "Hi, ";
        String mom = "mom.";
        hi+=mom;

        System.out.println(hi);

    }
}
```

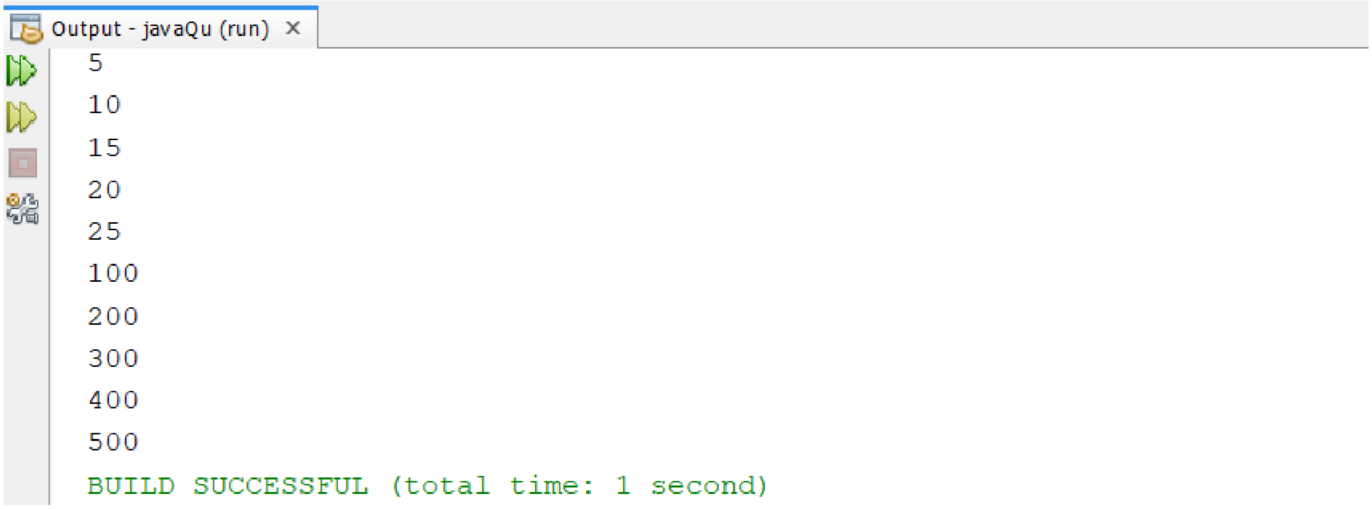
```
run:
Hi, mom.
BUILD SUCCESSFUL (total time: 0 seconds)
```

56.Write a java program to read a number of integers from a text file (ended by a string “end”) using the scanner class. Calculate and display the sum of the numbers.

Ans:



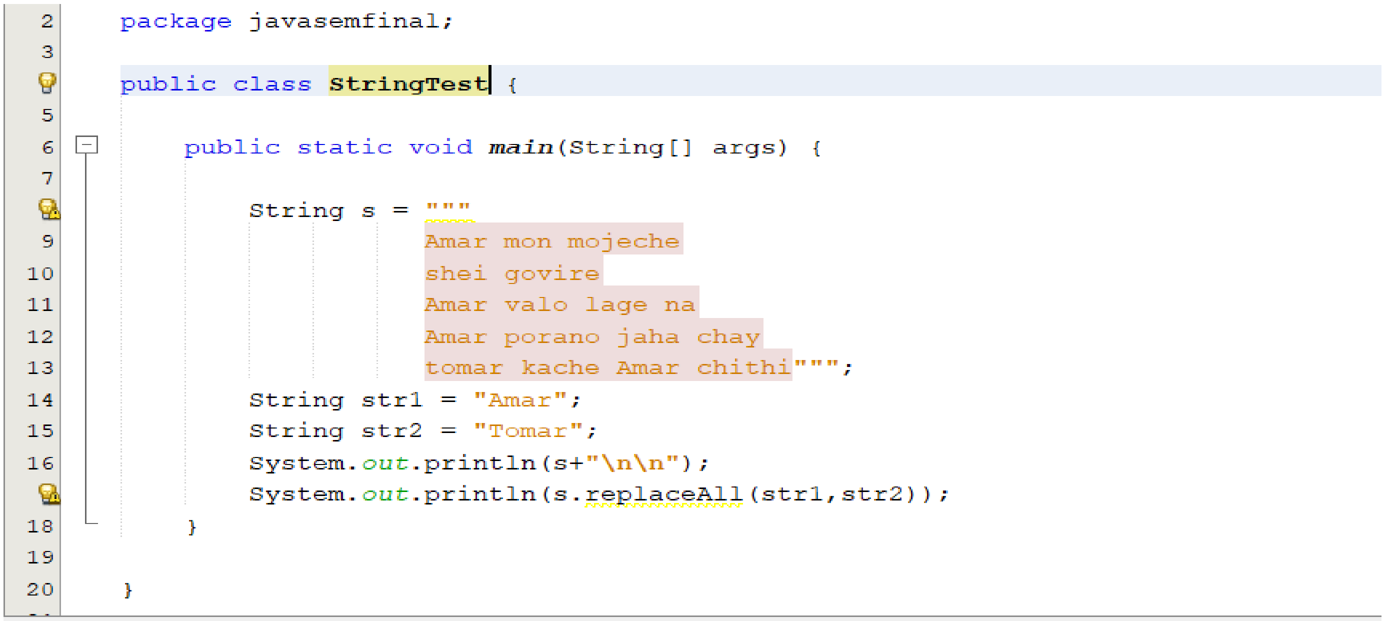
```
package javasemfinal;
import java.util.Scanner;
public class Why {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        String line;
        int sum = 0;
        int t = 1;
        try {
            while(t!=0){
                line = sc.nextLine();
                int p = Integer.parseInt(line);
                sum+=p;
            }
        } catch (NumberFormatException nfe) {
        }
        finally{
            System.out.println(sum);
        }
    }
}
```



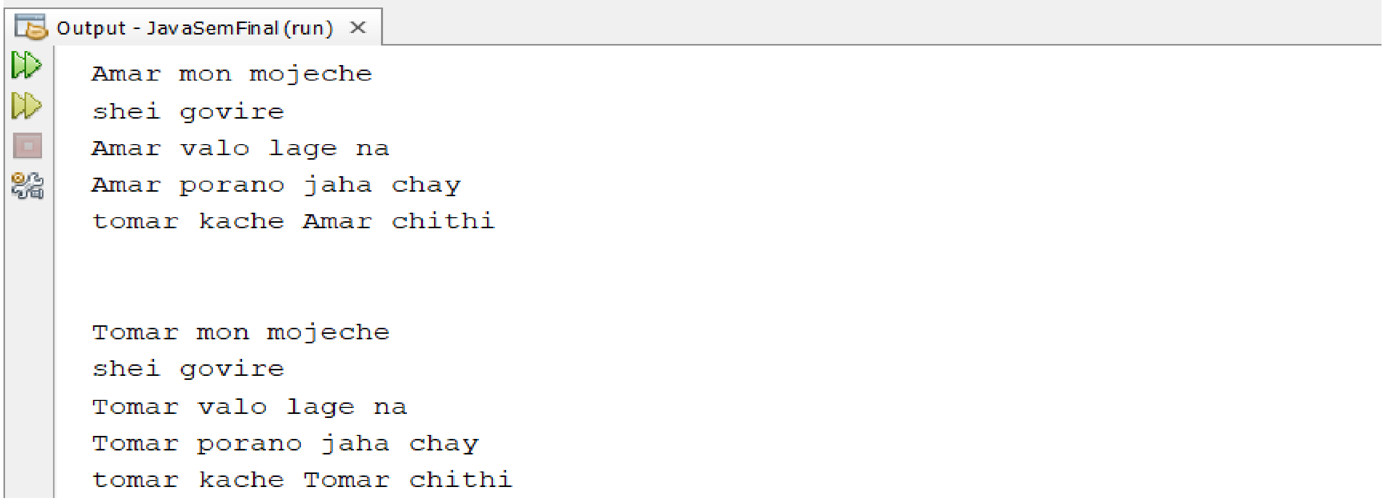
```
5
10
15
20
25
100
200
300
400
500
BUILD SUCCESSFUL (total time: 1 second)
```

57. Write a simple program to replace all the occurrences of a given word str1 in a string of several lines by another word str2.

Ans:



```java
package javasemfinal;

public class StringTest {

    public static void main(String[] args) {

        String s = """
                   Amar mon mojeche
                   shei govire
                   Amar valo lage na
                   Amar porano jaha chay
                   tomar kache Amar chithi""";
        String str1 = "Amar";
        String str2 = "Tomar";
        System.out.println(s+"\n\n");
        System.out.println(s.replaceAll(str1,str2));
    }

}
```



```
Amar mon mojeche
shei govire
Amar valo lage na
Amar porano jaha chay
tomar kache Amar chithi



Tomar mon mojeche
shei govire
Tomar valo lage na
Tomar porano jaha chay
tomar kache Tomar chithi
```