Java Syllabus:

Section 1 (Basic Java)

1) *Variable &Data Types*
2) *Loops*
3) *Control Statements*
4) *Methods*
5) *Constructor*

Section 2 (OOPS Concept)

1) *Inheritance*
2) *Polymorphism*
3) *Access Modifier*
4) *Abstract Class & Concrete Class*
5) *Interface & Implementation Class*
6) *Generalization*
7) *Casting*
8) *Encapsulation*

Section 3

1) *String Class*
2) *Array*
3) *Collection*
4) *Pattern Programs*
5) *Logical Programs*

What is Java?

Java is a class-based object-oriented programming language for building web anddesktop applications. It is the most popular programming language and the language of choice for Android programming.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) inthe year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is knownas a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Application:

1. *Desktop Applications such as acrobat reader, media player, antivirus, etc.*
2. *Web Applications such as irctc.co.in, javatpoint.com, etc.*
3. *Enterprise Applications such as banking applications.*
4. *Mobile*
5. *Embedded System*
6. *Smart Card*
7. *Robotics*
8. *Games, etc.*

Types of Java Applications

There are mainly 4 types of applications that can be created using Javaprogramming:

1) *Standalone Application*

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine.Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) *Web Application*

An application that runs on the server side and creates a dynamic page is called aweb application. Currently, Servlet, J SP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) *Enterprise Application*

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, E JB is used for creating enterprise applications.

4) *Mobile Application*

An application which is created for mobile devices is called a mobile application.Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java

1) *Java SE (Java Standard Edition)*

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topicslike OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) *Java EE (Java Enterprise Edition)*

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet,JSP, Web Services, EJB, J PA, etc.

3) *Java ME (Java Micro Edition)*

It is a micro platform that is dedicated to mobile applications

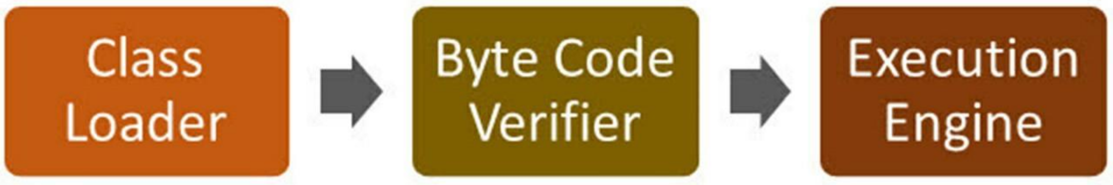
4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interfaceAPI.

Difference between JDK, JRE, and JVM

1) JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machinebecause it doesn't physically exist. It is a specification that provides a runtime environment in which Java byte code can be executed. It can also run those programs which are written in other languages and compiled to Java byte code.



JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from eachother. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

The JVM performs the following main tasks:

- *Loads code*
- *Verifies code*

- *Executes code*
- *Provides runtime environment*

2) *JRE*

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. TheJava Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besidesSun Micro Systems.

3) *JDK*

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is asoftware development environment which is used to develop Java applications andapplets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released byOracle Corporation:

- *Standard Edition Java Platform*
- *Enterprise Edition Java Platform*
- *Micro Edition Java Platform*

The JDK contains a private Java Virtual Machine (JVM) and a few other resourcessuch as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java

Application.

How Download and Install Java JDK?

Follow below links steps for download and install Java

1) Link for JDK Download: https://www.guru99.com/install-java.html

2) Link for Eclipse Download: https://www.guru99.com/install-eclipse-java.htm

---

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable.There are two types of data types in Java:

1. Primitive data types: The primitive data types include boolean, char, byte,short, int, long, float and double.

2. Non-primitive data types: The non-primitive data types include Class, Interfaces and Arrays

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation.These are the most basic data types available in Java language.

There are 8 types of primitive data types:

- *boolean data type*
- *byte data type*
- *char data type*
- *short data type*
- *int data type*

- *long data type*
- *float data type*
- *double data type*

Data Type	***Default Value***	***Default size***
boolean	*false*	*1 bit*
char	*'\u0000'*	*2 byte*
byte	*0*	*1 byte*
short	*0*	*2 byte*
int	*0*	*4 byte*
long	*0L*	*8 byte*
float	*0.0f*	*4 byte*
double	*0.0d*	*8 byte*

1) Boolean Data Type

The Boolean data type is used to store only two possible values:

true and false. This data type is used for simple flags that track true/falseconditions.

The Boolean data type specifies one bit of information, but its "size" can't be definedprecisely.

Example: Boolean one = false

2) Byte Data Type

The byte data type is an example of primitive data type. It is an 8-bit signed two'scomplement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than aninteger. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

3) Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range liesbetween -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A shortdata type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

4) Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range liesbetween - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive).

Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its defaultvalue is 0.

The int data type is generally used as a default data type for integral values unless if *there is no problem about memory.*

Example: int a = 100000, int b = -200000

5) Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between

-9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 - 1)(inclusive).

Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used whenyou need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

6) Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point. Its value rangeis unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

7) Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, suchas currency. Its default value is 0.0d.

Example: double d1 = 12.3

8) Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store *characters.*

Example: char letterA = 'A'

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit nextpage

---

What is a Variable in Java?

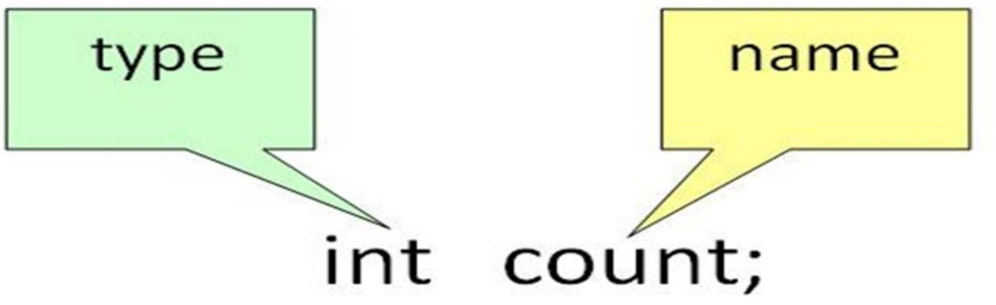
Variable in Java is a data container that stores the data values during Java program execution. Every variable is assigned data type which designates the type and quantity of value it can hold. Variable is a memory location name of the data. The Java variables have mainly three types: Local, Instance and Static.

In order to use a variable in a program you to need to perform 2 steps

1. *Variable Declaration*
2. *Variable Initialization*

Variable Declaration:

To declare a variable, you must specify the data type & give the variable a uniquename.



Examples:

int a;

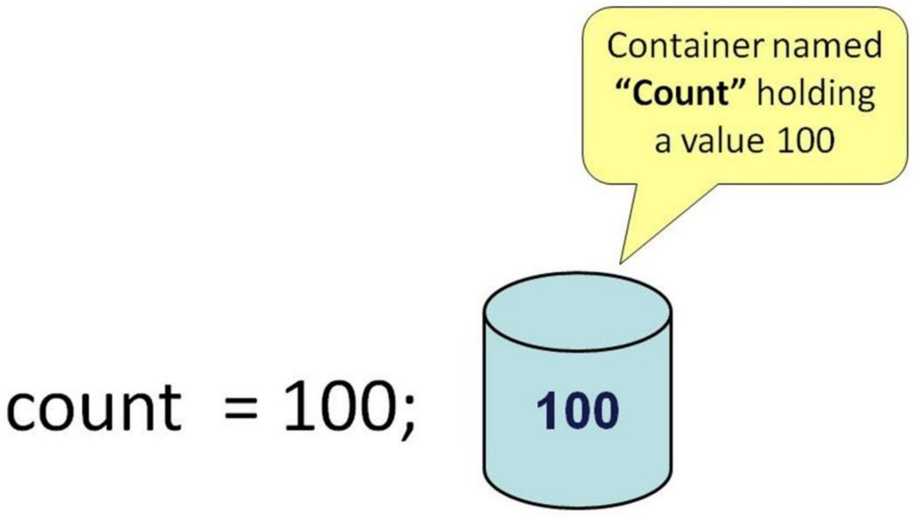
float pi;

double d;

char a;

Variable Initialization:

To initialize a variable, you must assign it a valid value.



Example of other Valid Initializations are

pi =3.14f;

do =20.22d;

a='v';

int a=2,b=4,c=6;

float pi=3.14f; double

do=20.22d;char

a='v';

Java Variable Types

In this section, we will learn about the various types of Java variables mentionedbelow.

- *Local variable*
- *Instance variable (Non-Static Variable)*
- *Static or Class variable*

Local Variables

These variables are declared inside the body of a method. These can be used withinthe same method where it is being initialized.

Some of the properties of a Local Variable include:

1. *Local variables are declared inside a method, constructor, or block.*
2. *No access modifiers for local variables.*
3. *These can be used only within the same block, method, or constructor whereit is initialized.*
4. *No default values after you have declared your local variable. You need to initialize your declared local variable.*
5. *It can't be defined by a static keyword.*

Given below is the program in which we have used local variables initialized within amethod of a class. As "height" is a local variable initialized with the calculate() method, the scope of this variable will be confined within the method.

```
public class local {

public void calculate() {

// initialized a local variable "height" int height = 0;

// incrementing the value of height
 height = height + 170;
System.out.println("height is: " + height + " cm");
}
public static void main(String args[]) {
// a is a reference used to call calculate() methodlocal a =
 new local();
a.calculate();
}
}
Output: height is: 170 cm
```

Instance Variables :

Instance variables are those variables that are declared inside a class. Unlike Local variables, these variables cannot be declared within a block, method, or constructor.

Example of Instance Variable

```
public class instance {
```

```
// Declaring instance variables public int rollNum;public
String name;
 public int totalMarks;
public int number;
public static void main(String[] args) {
// created object
instance in = new instance();
in.rollNum = 95;
in.name = "Bharat";
in.totalMarks = 480;
// printing the created objects System.out.println(in.rollNum);
System.out.println(in.name); System.out.println(in.totalMarks);

/*
*         we did not assign the value to number so it
*         will print '0' by default
*/ System.out.println(in.number);
}
}
```

Output:

```
95
Bharat 480
0
```

Static or Class Variable

Unlike the Local and Instance variable (where we cannot use static), we have anothervariable type which is declared as static and is known as "Static or Class variable"

Example

```
public class StaticVariable {

// radius is declared as private static
 private static int radius;

// pi is a constant of type double declared as staticprivate
 static final double pi = 3.14;

public static void main(String[] args) {

// assigning value of radius
 radius = 3;

// calculating and printing circumference System.out.println("Circumference of
 a circle is: " + 2*pi*radius);
}
}
```

Output: Circumference of a circle is: 18.84

What is Class ?

Class:-

>>Class is a combination of methods and variables. It is just a blueprint ortemplate.

1. *Class is used to store static as well as non-static members inside it.*
2. *It does not occupy memory.*
3. *Syntax of class:-*
   Access modifier Class Classname .

Syntax :

```
class <class_name>
{
field;
method;
}
```

What is Object in Java?

=Object is an instance of a class.

=We can creat object by using "NEW" keyword

= An object in OOPS is nothing but a self-contained component which consists ofmethods and properties to make a particular type of data useful.

= For example color name, table, bag, barking. When you send a message to an object, you are asking the object to invoke or execute one of its methods as definedin the class.

From a programming point of view, an object in OOPS can include a data structure, avariable, or a function. It has a memory location allocated.

Syntax :

```
        Classname        Reference variable     =   new        Classname
Ex.
        Animal  a   = new   Animal ();
```

What is a method in Java?

1) A method is a block of code or collection of statements or a set of codegrouped together to perform a certain task or operation.

2) It is used to achieve the reusability of code. We write a method once and useit many times. We do not require to write code again and again.

3) It also provides the easy modification and readability of code, just by addingor removing a chunk of code.

4) The method is executed only when we call or invoke it.

5) The most important method in Java is the main() method

Syntax:

Access Specifier returnType nameOfMethod (Parameter List){

// method body

}

The syntax shown above includes –

•Access Specifier − It defines the access type of the method and it is optional to use.

•returnType − Method may return a value.

• nameOfMethod − This is the method name. The method signature consists of the method name and the parameter list.

• Parameter List − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

• method body − The method body defines what the method does with the statements.

What is Access Modifier?

You might've come across public, private and protected keywords while practicing any Java programs, these are called the Access Modifiers. As the name suggests, Access Modifiers in Java helps to restrict the scope of a class, constructor, variable,method or data member.

Access modifiers can be specified separately for a class, constructors, fields, andmethods. They are also referred as Java access specifiers, but the correct name isJava access modifiers.

Types of Access Modifier

There are four access modifiers keywords in Java and they are:

Default Access Modifier

Private Access Modifier

Public Access Modifier

Protected Access Modifier

1) Default: Whenever a specific access level is not specified, then it is assumedto be 'default'. The scope of the default level is within the package.

2) Private: When an entity is private, then this entity cannot be accessed outsidethe class. A private entity can only be accessible from within the class.

3) Public: This is the most common access level and whenever the publicaccess specifier is used with an entity, which particular entity is accessible throughout from within or outside the class, within or outside the package, etc.

4) Protected: The protected access level has a scope that is within the package.A protected entity is also accessible outside the package through inherited class or child class.

Java Control Statements:

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, java provides statements that can be used to control the flow of Java code. Such statements arecalled control flow statements. It is one of the fundamental features of Java, whichprovides a smooth flow of program.

Java provides three types of control flow statements.

1. *Decision Making statements*
   - = *if statements*
   - = *switch statement*
2. *Loop statements*
   - = *do while loop*
   - = *while loop*
   - = *for loop*
   - = *for-each loop*
3. *Jump statements*
   - = *break statement*
   - = *continue statement*

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.

There are two types of decision-making statements in Java, i.e., If statement andswitch statement.

1) If Statement:

In Java, the if statement is used to evaluate a condition. The control of the programis diverted depending upon the specific condition. The condition of the If statementgives a Boolean value, either true or false. In Java, there are four types of if- statements given below.

1. *Simple if statement*
2. *if-else statement*
3. *if-else-if ladder*
4. *Nested if-statement*

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code ifthe expression evaluates to true.

Syntax

of if statement is given below,

```
if(condition) {
statement 1;          //executes when condition is true
}
```

Consider the following example in which we have used the if statement in the javacode.

Example:

```
public class Student
{
public static void main(String[] args) {int
x = 10;
 int y = 12;
 if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

Output: x + y is greater than 20

2) if-else statement

The "if-else statement" is an extension to the if-statement, which uses another blockof code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

```
Syntax: if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

Example:

```
public class Student
{
public static void main(String[] args) {int
x = 10;
 int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than              10");
}
else {
System.out.println("x + y is greater than 20");
}
}
}
```

Output: x + y is greater than 20

3)   if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements thatcreate a decision tree where the program may enter in the block of code where thecondition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
if(condition 1) {
statement 1;          //executes when condition 1 is true
}
else if(condition 2) {
statement 2;          //executes when condition 2 is true
}
else {
statement 2;           //executes when all the conditions are false
}
```

Example:

```
public  class  Student
{
public static void main(String[] args) {
 String city = "Delhi";

if(city == "Meerut") {
 System.out.println("city  is  meerut");
}
else if (city == "Noida") {
 System.out.println("city  is  noida");
}
else if(city == "Agra") {
System.out.println("city  is  agra");
}
else {  System.out.println(city);
}
}
}
```

Output: Delhi

===

4. Nested if-statement

In nested if-statements, the if statement can contain a if or if-else statement insideanother if or else-if statement.

Syntax of Nested if-statement is given below.

```
if (condition 1) {
statement 1; //executes when condition 1 is trueif
(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when condition 2 is false
}
}
```

Example:

```
public class Student {
public static void main(String[] args) { String address = "Delhi, India";

if(address.endsWith("India")) { if(address.contains("Meerut"))
{ System.out.println("Your city is Meerut");
}
else if(address.contains("Noida")) { System.out.println("Your city is Noida");
}
else { System.out.println(address.split(",")[0]);
}
}
else {
System.out.println("You are not living in India");
}
}
}
```

Output: Delhi

Switch Statement:

In Java,Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of theprogram.

Points to be noted about switch statement:

-The case variables can be int, short, byte, char, or enumeration. String type isalso supported since version 7 of Java

-Cases cannot be duplicate

-Default statement is executed when any of the case doesn't match the valueof expression. It is optional.

-Break statement terminates the switch block when the condition issatisfied. It is optional, if not used, next case is executed.

-While using switch statements, we must notice that the case expression willbe of the same type as the variable. However, it will also be a constant value.

The syntax to use the switch statement is given below.

```
switch (expression){
case value1:
 statement1;
break;
.
.
.
case valueN:
statementN;
break; default:
default statement;
}
```

Example:

```
public class Student implements Cloneable
{
public static void main(String[] args) {int
num = 2;
switch (num){
case 0:
System.out.println("number  is  0");
break;
case 1:
System.out.println("number  is  1");
break;
default:
System.out.println(num);
}
}
}
```

Output: 2

Constructor:

CONSTRUCTOR is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object. It can be used to initialize the objects to desired values or default values at the time of object creation.It is not mandatory for the coder to write a constructor for a class.

If no user-defined constructor is provided for a class, compiler initializes membervariables to its default values.

Note: It is called constructor because it constructs the values at the time of objectcreation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating a Java Constructor

1. *It has the same name as the class*
2. *It should not return a value not even void*
3. *A Java constructor cannot be abstract, static, final, and synchronized*
4. *We can have private, protected, public or default constructor in Java.*

Note: Constructor has the same name as that of the class and does not have anyreturn type.

For example, class Test {

```
  Test() {
    // constructor body
             }
}
```

Here, Test() is a constructor.

It has the same name as that of the class and doesn't have a return type.

==
=

Constructor Example:

```
class Main {
 private String name;
  // constructor
  Main() {
    System.out.println("Constructor Called:");
  name = "Programiz";
 }
 public static void main(String[] args) {                    // constructor is invoked while
    // creating an object of the Main classMain
    obj = new Main();
    System.out.println("The name is " + obj.name);
    }
 }
```

Output: *Constructor Called:*

The name is Programiz

In the above example, we have created a constructor named Main(). Inside theconstructor, we are initializing the value of the name variable.

Notice the statement of creating an object of the Main class.Main obj

= new Main();

Here, when the object is created, the Main() constructor is called. And, the value ofthe name variable is initialized.

Hence, the program prints the value of the name variables as Programiz.

==
==============

Types of Java constructors

There are two types of constructors in Java:

1. *Default constructor*
2. *No-arg constructor*
3. *Parameterized constructor*

1) Default Constructor:

A) If we do not create any constructor, the Java compiler automatically create ano-arg constructor during the execution of the program. This constructor is calleddefault constructor.

B) You would not find this constructor in your source code means in .java file , itwould be inserted into code during compilation and exists in .class file. C) This Constructor not visible to Programmer.

2) No-arg Constructor:

A) Constructor with no arguments called no-arg Constructor. Or If a constructordoes not accept any parameters, it is known as a no-argument constructor.

B) The signature is same as default constructor, however body can have anycode unlike default constructor where body of the constructor empty.

C) If you write public Demo () {} in your class Demo it cannot be called default constructor since you have written the code of it.

D) This Constructor Visible to Programmer.

Example:

```
Class Demo
{
Public Demo()
{
System.out.println (“This is a no argument constructor”);
}
Public static void main (String[] args)
{
New Demo();
}
}
```

Output: *This is a no argument constructor.*

3) *Parameterized Constructor:*

A) Constructor with arguments (Parameters) is known as Parameterized Constructor.

B) We can pass multiple arguments same time. class Main {String

```
 languages;
 // constructor accepting single value
  Main(String lang) {
languages = lang;
   System.out.println(languages + " Programming Language");
 }
 public static void main(String[] args) {
   // call constructor by passing a single valueMain
   obj1 = new Main("Java");
   Main obj2 = new Main("Python");Main
   obj3 = new Main("C");
 }
}
```

Output:

```
Java Programming Language
Python Programming LanguageC
Programming Language
```

Difference between Method and Constructor

Constructor	Method
1)Constructor is used to initialize an object	1) Method is used to exhibits functionality of an object
2) No need to call if not called then there will be default constructor.	2)Need to call(if not called then no run)

3)Constructor does not return any value not even void	*3) Method has return type*
4)Name of the constructor must besame with the name of the class	*4)In methods no such requirements*
5)We cannot override Constructor	*5) Method overriding possible.*
6) Constructors are not inherited	*6) Non-static methods are inherited.*
7) A constructor can be called usingnew keyword	*7) method can be called using object (Non static methods) or class reference (Static methods) or byname*
8) A constructor can call another constructor using this() or super() Keyword	*8) Method can call other method directly or using class or object reference.*
9) A constructor cannot be static,abstract , final, native or synchronized	*9) Method can be static, abstract, finaland native or Synchronized.*

Section-2: (OOPs Concept)

1) *Inheritance*
2) *Polymorphism*
3) *Abstraction*
4) *Encapsulation*

1) Inheritance:

A) *Inheritance it is oops principle where one class acquired the property of otherclass.*

B) *The class from which property acquired or inherits is called Super or ParentClass.*

C) *The class in which property are delivered is called Sub Class or Child Class.*

D) *For Inheritance extends keyword used.*

Types of Inheritance:

A) *Single Level Inheritance*

B) *Multi Level Inheritance*

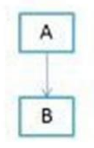
C) *Multiple Inheritance*

D) *Hierarchical Inheritance*

E) *Hybrid Inheritance*

1) Single Level Inheritance

Single Level inheritance is easy to understand. When a class extends another oneclass only then we call it a single Level inheritance. The below flow diagram showsthat class B extends only one class which is A. Here A is a parent class of B and B would be a child class of A.



(a) Single Inheritance

Example:

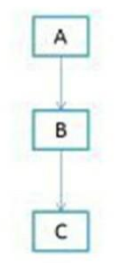
```
Class A {
 public void methodA() { System.out.println("Base
   class method");
  }
}
Class B extends A {
public void methodB() {
    System.out.println("Child  class  method");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA();                          //calling super class method
    obj.methodB();                          //calling local method
  }
}
```

==
========
========

2) Multi-level Inheritance

A) In multi-Level inheritance one sub class acquired the property of super orParent Class which also acquired property of another sub class.

B) In which minimum two sub classes are required to performing multi-level inheritance.



(d) Multilevel Inheritance

Example:

```
Class A {
  public void methodA() { System.out.println("Class
    A  method");
  }
}
Class B extends A
{
public void methodB()
{
System.out.println("class  B  method");
}
}
Class C extends B
{
  public void methodC()
  {
    System.out.println("class  C  method");
  }
  public static void main(String args[])
  {
    C obj = new C();
    obj.methodA();          //calling grand parent class method
    obj.methodB();          //calling parent class method
    obj.methodC();          //calling local method
  }
}
```

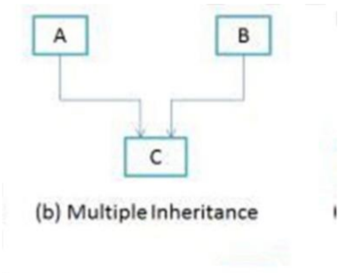
***3)** Multiple Inheritance:*

A) When one class extends more than one classes then this is called multiple inheritance.

For example:

Class C extends class A and B then this type of inheritance is known asmultiple inheritance.

B) Multiple inheritance not supported in Java because of "Diamond Ambiguity" Problem.



(b) Multiple Inheritance

C) In above diagram Class C extends Both Class A and Class B and inboth class have same method name display ().

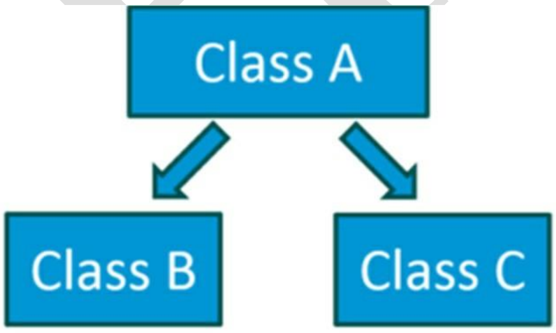
D) Now Java Compiler cannot decide, which display () method it shouldinherit.

E) So prevent such situation multiple inheritance not supported in java.

==
========
==============

4) Hierarchical Inheritance

When a class has more than one child classes (sub classes) or in other words,more than one child classes have the same parent class, then such kind of inheritance is known as hierarchical.



If we talk about the flowchart, Class B and C are the child classes which areinheriting from the parent class i.e Class A.

Let's see the syntax for hierarchical inheritance in Java:Class A {

}

Class B extends A {

}

Class C extends A {

}

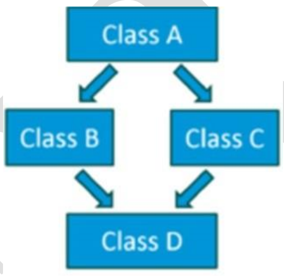
===

=============

5) Hybrid Inheritance:

Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance. Since multiple inheritance is not supported in Java as it leads to ambiguity, so this type of inheritance can only be achieved through the use of theinterfaces.

Diagram for reference :



If we talk about the flowchart, class A is a parent class for class B and C, whereasClass B and C are the parent class of D which is the only child class of B and C.

2) Polymorphism

Poly = many and morphism = form

>Polymorphism is the concept where an object behaves differently in differentsituations.

>When one object or element performs different behaviours at different stages of life cycle is called Polymorphism.

>Polymorphism means taking many forms, where 'poly' means many and 'morph' means forms.

It is the ability of a variable, function or object to take on multiple forms. In other words, polymorphism allows you define one interface or method and have multiple implementations.

Polymorphism in Java is of two types:

1. *Compile time polymorphism*
2. *Run time polymorphism*

1) Compile Time Polymorphism

>In compile time polymorphism method declaration get binded to its definition (body)at the time of compilation based on Parameter or Arguments it is called "Compile Time Polymorphism".

>We can achieve by using method overloading

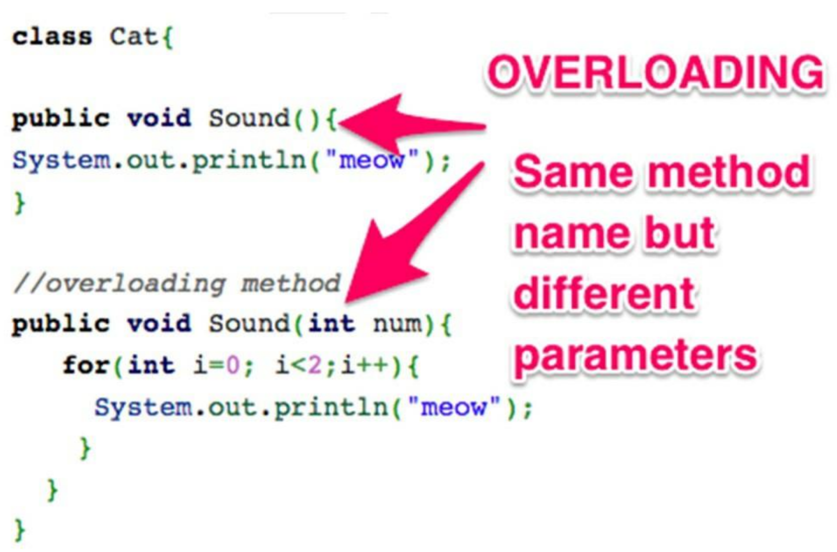
>Method Overloading will be performing in

Same Class only.

>Same Method Name

> but different Arguments or Parameter

Example :



```
class Cat{

public void Sound(){
System.out.println("meow");
}

//overloading method
public void Sound(int num){
   for(int i=0; i<2;i++){
     System.out.println("meow");
    }
  }
}
```

There are 3 Ways of Method Overloading:

1. Change number of parameters

2.Data type of parameters 3.Order of parameters

1) Change Number of Parameters

By changing the number of parameters you can achieve method overloading.For example:

add (int , int)   // two parameters

add (int, int, int)   // three parameters Practical Example:

```
class MethodOverloadingExample {public
     void show(String s)
   {
     System.out.println(s);
```

```
    }
      public void show(int i , String s)
    {
      System.out.println(i + " " + s);
    }
     public static void main(String args[])
    {
      MethodOverloadingExample ol = new MethodOverloadingExample ();
            ol.show ("Be in Present");
            ol.show (10, "Be in Present");
    }
}
```

Output:

```
"Be in Present" 10
"Be in Present"
```

In the above example, show () method is overloaded based on the number ofparameters. Both show () methods have different parameters. One is accepting String type whileother is accepting int and String parameters.

2. Data type of parameters:

Method overloading can be achieved by changing the data type of parameters.For example:

```
add (int , int)
add (int, float)
```

Practical Example:

```
class MethodOverloadingExample {
    public void show(String s)
 {
    System.out.println(s);
 }
```

```
  public void show(int i)
{
  System.out.println (i + " " + s);
}
  public static void main(String args[])
{
  MethodOverloadingExample ol = new MethodOverloadingExample ();ol.show
              ("Be in Present");
              ol.show (10);
}
}
```

Output:

```
"Be in Present"
10
```

In the above example method overloading is done by changing the data type of theparameters. One show () method is accepting String parameter while other show () method isaccepting int parameter.

3. Order of parameters:

Method overloading can be done by changing the order of parameters.For example:

add(float , int)

add(int, float)

Practical Example

```
class MethodOverloadingExample {public
    void show(String s , int i)
{
    System.out.println(s + " " + i);
}
    public void show(int i , String s)
{
    System.out.println (i + " " + s);
 }
    public static void main(String args[])
{
  MethodOverloadingExample ol = new MethodOverloadingExample ();
            ol.show("Be in Present" ,10);
            ol.show(15 , "Alive is Awesome");
 }
}
```

Output:

```
"Be in Present" 10
10 "Be in Present"
```

In the above example both show () methods have different sequence of data type ofparameters. Since the first show () method has parameters (int, String) and second show ()method has parameters (String, int).

The above case is a valid method overloading.

---

2) Run Time Polymorphism (Dynamic Polymorphism)

>When method declaration binded with its definition (Body) at the time of execution(Run) based on object and instance it is called as "Run Time Polymorphism".

>"Method Overriding" is example of Run Time Polymorphism.

We can achieve run time polymorphism by using method overridingMethod Overriding Means

There is same method name , same args , IS A inheritance and class is differentExample:

```
public class ParentClass {
 // Overridden method
  public void printData()
   {
    System.out.println("In this method we are showing the details of Parent class");
   }
 }
 public class ChildClass extends ParentClass
{
     //Overriding method
 public void printData()
    {
         System.out.println("In this method we are showing the details of child class");
         System.out.println("We are overriding the method");
    }
   public static void main( String args[]) {
        ParentClass parentObject = new ChildClass ();                    //This will call the child
class method
                parentObject.printData ();
   }
}
```

Output:

```
In this method we are showing the details of child class We are overriding themethod
```

Important Questions:

Q1. What is difference between Overloading and Overriding?

Method Overloading	***Method Overriding***
1) Method overloading is a compile time polymorphism.	*1. Method overriding is a run time polymorphism.*
2. It helps to raise the readability of the program.	*2. While it is used to grant the specific implementation of the method which is already provided by its parent class or super class.*

3. It is occur within the class.	*3. While it is performed in two classeswith inheritance relationship.*
4. Method overloading may not require inheritance.	*4. While method overriding always needs inheritance.*
5. In this, methods must have same name and different signature/Declaration.	*5.While in this, methods must have same name and same signature/Declaration but different Definition(Body)*
6. In method overloading, return type can or cannot be same, but we must have to change the parameter.	*6. While in this, return type must be same or co-variant.*

Q2) Can we overload Main Method?

>So can we overload main method in Java?Answer is

Yes, We can.

>Execution of a Java program starts from main () method.

In fact we should say that Execution of a Java program starts from a main () methodwhose actual syntax is "public static void main (String [] args)".

>We can write multiple valid overloaded main method in a class but JVM alwayslooks for a main method with syntax as "public static void main(String[] args)".

If it finds, it will start execution from there otherwise will give error stating "Error:Could not find or load main".

Practical Example:

```
 public class OverloadedMainMethod {
        // overloaded main method
public static void main(String str)
        {
        System.out.println ("public static void main (String str)");
        }
        // Overloaded main method
public static int main(int str)
        {
        System.out.println ("public static void main (String str)");
        return 1;
        }
        // overloaded main method
        public void main(String str, int val)
        {
                System.out.println ("public void main (String str, int val)");
        }
// Main method from where execution startspublic
static void main(String[] args)
{
        System.out.println ("public static void main (String [] args)");
 // Calling other main method
          main ("Bharat");
          main (10);
        new OverloadedMainMethod ().main("Bharat", 10);
        }
}
```

2) Can we overload static Method?

Yes, we can overload the static method. A class can have more than one static method with the same name, but different in input parameters. You can overload the static method by changing the number of arguments or by changing the type of arguments.

Example:

```
public class MethodOverloadingOfStaticMethod
{
 public static void show(String name)
```

```
    {
        System.out.println ("Name of person = "+name);
    }
    public static void show(int age)
    {
        System.out.println ("Person age is = "+ age);
    }
    public static void main (String [] args) {
        // If user providing parameter of String type then first method called
        MethodOverloadingOfStaticMethod.Show ("Ram");
        // If user providing parameter of int type then second method called
        MethodOverloadingOfStaticMethod.Show (18);
    }
}
```

```
Output: Name of person = Ram
Person age is = 18
```

---

3) Can we override Final Methods?

>A method declared with the final keyword, and then it is known as the final method.The final method can't be overridden.

>A final method declared in the Parent class cannot be overridden by a child class. Ifwe try to override the final method, the compiler will throw an exception at compile time. Because if we are declaring any method with the final keyword, we are indicating the JVM to provide some special attention and make sure no one can override it.

>final is a keyword in java used for restricting some functionalities. If we aredeclaring any method with final keyword it means we can't change its implementation in child class. So, Java doesn't permit to override any final method.

Example:

```
public class Parent
{
    // final method in base class
   final void showData()
    {
        System.out.println("This is Static method of Parent class");
    }
}
public class Child extends Parent
{
    // final method in drived class
   final void showData()
    {
  System.out.println ("This is Static method of child class");
    }
}
public class MainClass {
      public static void main(String[] args) {
        Parent parent = new Child();
        parent.showData();
    }
}
```

Output: The above example will show compilation error

Q.4 Can we override Static Method?

>We cannot override Static Method.

>If you have a static method in Parent class and you defining a static method with same signature as a static method in Parent class. In this case, the Child class method would act differently. It is known as method hiding.

Practical Example:

```
 public class Parent  {
// Static method in base class
    static void showData()
    {
        System.out.println("This is Static method of Parent class");
    }
}
public class Child extends Parent
{
    // static method in derived class
   static void showData()
    {
        System.out.println("This is Static method of child class");
    }
}
public class MainClass {
    public static void main(String[] args)
    {
        Parent parent = new Child ();
              parent.showData ();
    }
}
```

Output: This is Static method of Parent class

As you know Static method can directly be called with class name. In this example we are calling showData() method by reference of Parent class then compilerconverts.

3) ABSTRACTION

= It means hiding the implementation and showing the service that has been

provided.

= Ex.

Car , ATM

= There are 2 ways to achieve abstraction

= By Abstract class (0-100%)

= By Interface (100%)

= It contains abstract & concrete method.

= Abstract method don't have body and concrete method have body .

= As both methods are used in it, it is partially abstracted hence abstraction can beachieved from 0 to 100 % .

= We don't create object of abstract class, we create reference of it .

= We use extends keyword for inheritance .

= overriding can be done here.

= when method abstract then class can also be abstract .

= But there can be abstract as well as Concrete method in abstract class.

By Abstract Class (0 - 100 %)

Example :

```
  abstract class Bike{
 abstract void run();
}
class Honda4 extends Bike{
void run(){
System.out.println("running  safely");
}
public static void main(String[]args){Bike
 obj = new Honda4();
 obj.run();
}
}
```

Output: *running safely*

By Interface (100 %)

= Interface contains only abstract method so abstraction can be achieved 100% .

= we don't create object ,we create reference .

= by interface we can achieve multiple inheritance .

= Here we use implement keyword .

= variable in interface is public static final .Example :

```
Public Interface I1
{
   Void show();                              // abstract method
}
Public Interface I2
{
    Void display();
}
Public class Implement I1,I2
{
  Public void show ()
{
System.out.println("1");
}
  Public void display ()
{
System.out.println("2");
}
Public static void main (string[]args){
Test a = new test ();
a.Show ();
a.display();
}
}
```

Difference between abstract class and interface

Abstract class	**interface**
Abstract class can have abstract and non-abstract methods.	*Interface can have only abstract methods. Since Java 8, it can have default and static methods also.*
Abstract class doesn't support multiple inheritance	*Interface supports multiple inheritance.*
Abstract class can have final, non- final, static and non-static variables.	*Interface has only static and final variables.*
Abstract class can provide the implementation of interface.	*Interface can't provide the implementation of abstract class.*
The abstract keyword is used to declare abstract class.	*The interface keyword is used to declare interface.*
An abstract class can extend another Java class and implement multiple Java interfaces.	*An interface can extend another Java interface only.*
An abstract class can be extended using keyword "extends".	*An interface can be implemented using keyword "implements".*
A Java abstract class can have classmembers like private, protected, etc.	*Members of a Java interface are public by default.*

Example:
public abstract class Shape{ public abstract void draw();
}
	Example:
	public interface Drawable{ void draw();
	}

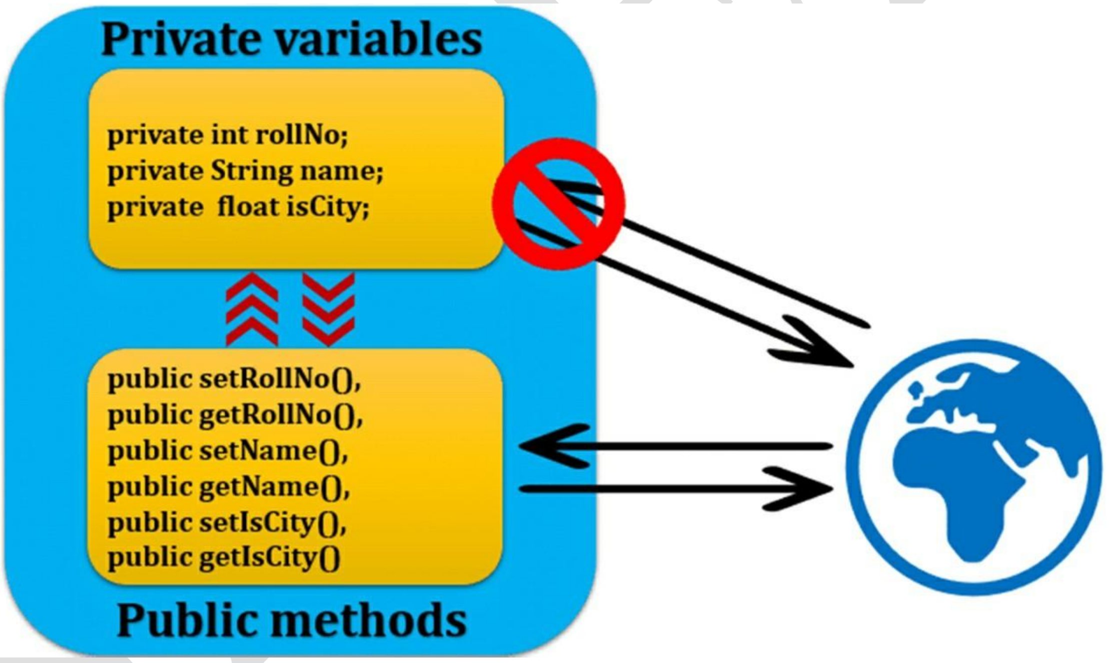
We achieve (0–100 %) abstraction	*We achieve (100 %) abstraction .*

4) Encapsulation

It is a mechanism of wrapping methods and variables in a single unit .

We can achieve an encapsulation by two ways:-

1. *By keeping class as private i.e. data hiding.*
2. *By declaring public getter and setter method for viewing and modifying the data.It is used for banking transactions as data hiding is done for security purpose. Thedisadvantage of Encapsulation is every time we have to declare getter and settermethod due to which the length of the code increases and it affects theperformance of an application leading it to work slowly.*



Example:

public class Student

{

// For encapsulation all variables should be private

```
  private int rollNo;
  private String name;
  private String className;
   //You should provide setter and getters for variablespublic
int getRollNo (){
        return rollNo;
  }
public String getName(){
        return name;
}
public String getClassName(){
       return className;
 }
 public void setRollNo (int value){
      rollNo = value;
 }
 public void setName (String value){
        name = value;
    }
 public void setClassName (String value){
       className = value;
    }
}
public class EncapsulationExample { public
    static void main(String args[])
    {
         Student obj = new Student ();
                obj. setRollNo (1);
                obj. setName ("RAVI");
        obj. setClassName ("MCA");
    System.out.println ("Student RollNo: " + obj.getRollNo ()); System.out.println
    ("Student Name: " + obj.getName ()); System.out.println ("Student Class: " +
    obj.getClassName ());
  }
}
```

```
Output:
Student RollNo: 1
Student Name: RAVI
Student Class: MCA
```

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string.

For example:

```
1.      char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2.      String s=new String(ch);is
same as:
1.      String s="java";
```

Java String class provides a lot of methods to perform operations on stringssuch as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

CharSequence Interface

The CharSequence interface is used to represent the sequence of characters.String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.

The Java String is immutable which means it cannot be changed. Whenever *we change any string, a new instance is created. For mutable strings, you can useStringBuffer and StringBuilder classes.*

We will discuss immutable string later. Let's first understand what String inJava is and how to create the String object.

What is String in Java?

Generally, String is a sequence of characters. But in Java, string is an objectthat represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. *By string literal*
2. *By new keyword*

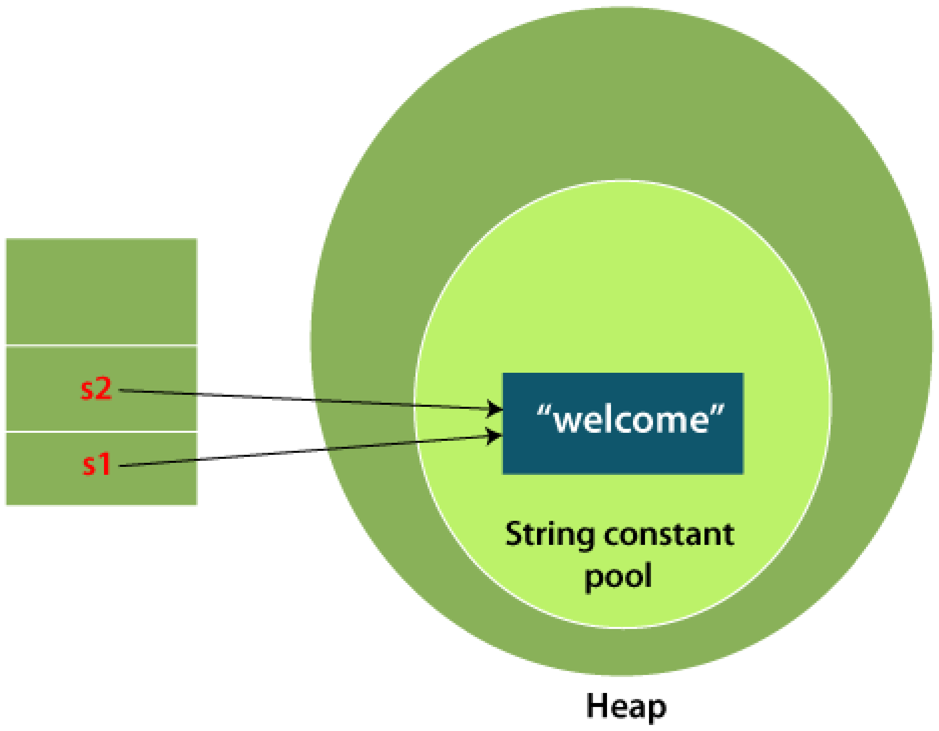
1) String Literal

Java String literal is created by using double quotes. For Example:

```
1.    String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool"first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
1.    String  s1="Welcome";
2.    String s2="Welcome";        //It doesn't create a new instance
```



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created ifit exists already in the string constant pool).

2) *By new keyword*

1. String s=new String("Welcome"); //creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heapmemory, and the literal "Welcome" will be placed in the string constant pool. Thevariable s will refer to the object in a heap (non-pool).

Java String Example

StringExample.java

```
public class StringExample{
public static void main(String args[]){
String s1="java";          //creating string by Java string literal
```

```
        char   ch[]={'s','t','r','i','n','g','s'};

        String s2=new String(ch);        //converting char array to string String

        s3=new  String("example");        //creating Java string by new
keyword

        System.out.println(s1);

        System.out.println(s2);

        System.out.println(s3);

        }
```

Output:

```
java

strings

example
```

The above code, converts a char array into a String object. And displays theString objects s1, s2, and s3 on console using println() method.

Java String class methods

The java.lang.String class provides many useful methods to performoperations on sequence of char values.

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
5	String substring(int beginIndex)	It returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
10	boolean equals(Object another)	It checks the equality of string with the given object.
11	boolean isEmpty()	It checks if string is empty.
12	String concat(String str)	It concatenates the specified string.
13	String replace(char old, char new)	It replaces all occurrences of the specified char value.

14	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
16	String[] split(String regex)	It returns a split string matching regex.
17	String[] split(String regex, int limit)	It returns a split string matching regex and limit.
18	String intern()	It returns an interned string.
19	int indexOf(int ch)	It returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
21	int indexOf(String substring)	It returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
23	String toLowerCase()	It returns a string in lowercase.
24	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
25	String toUpperCase()	It returns a string in uppercase.
26	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
27	String trim()	It removes beginning and ending spaces of this string.
28	static String valueOf(int value)	It converts given type into string. It is an overloaded method.

Immutable String in Java

A String is an unavoidable type of variable while writing any application program. String references are used to store various attributes like username,password, etc. In Java, String objects are immutable. Immutable simply meansunmodifiable or unchangeable.

Once String object is created its data or state can't be changed but a newString object is created.

Let's try to understand the concept of immutability by the example given below:

```
class Testimmutablestring{

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

   s="Sachin";

   s.concat(" Tendulkar");          //concat() method appends the string at the end

   System.out.println(s);           //will print Sachin because strings are immutable
objects

 }

}
```

Output:

```
Sachin
```

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with Sachin Tendulkar. That is why String isknown as immutable.

As you can see in the above figure that two objects are created but sreference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitly assign it to the reference variable, it will refer to "SachinTendulkar" object.

For example:

```
class Testimmutablestring1{
```

```
public static void main(String args[]){String
  s="Sachin";
  s=s.concat(" Tendulkar");
  System.out.println(s);
}
   }
```

Output:

```
Sachin Tendulkar
```

In such a case, s points to the "Sachin Tendulkar". Please notice that stillSachin object is not modified.

Why String objects are immutable in Java?

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the valueof the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

Following are some features of String which makes String objects immutable.

1. ClassLoader:

A ClassLoader in Java uses a String object as an argument. Consider, if theString object is modifiable, the value might be changed and the class that is supposed to be loaded might be different.

To avoid this kind of misinterpretation, String is immutable.

2. Thread Safe:

As the String object is immutable we don't have to take care of the synchronization that is required while sharing an object across multiple threads.

3. Security:

As we have seen in class loading, immutable String objects avoid further errors by loading the correct class. This leads to making the application program more secure. Consider an example of banking software. The username and password cannot be modified by any intruder because String objects are immutable.This can make the application program more secure.

4. Heap Space:

The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the *new object. This feature allows Java to use the heap space efficiently.*

Why String class is Final in Java?

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

STRING COMPARE

1) By Using equals() Method

The String class equals() method compares the original content of the string. Itcompares values of string for equality. String class provides the following twomethods:

public boolean equals(Object another) compares this string to the specified object.public

boolean equalsIgnoreCase(String another) compares this string to another string, ignoring case.

EX.

```
class Teststringcomparison1{
 public  static  void  main(String  args[]){
   String s1="Sachin";
   String s2="Sachin";
   String   s3=new   String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));              //true
   System.out.println(s1.equals(s3));              //true
   System.out.println(s1.equals(s4));  //false
 }
}
```

Output:

```
true
true
```

false

In the above code, two strings are compared using equals() method of String class.And the result is printed as boolean values, true or false.

```
class Teststringcomparison2{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";

   System.out.println(s1.equals(s2));//false System.out.println(s1.equalsIgnoreCase(s2));//true
 }
}
```

Output:

```
false
true
```

In the above program, the methods of String class are used. The equals() methodreturns true if String objects are matching and both strings are of same case. equalsIgnoreCase() returns true regardless of cases of strings.

Click here for more about equals() method

---

2) By Using == operator

The == operator compares references not values.

```
class Teststringcomparison3{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
```

```
  String s3=new String("Sachin");
  System.out.println(s1==s2);         //true (because both refer to same instance)
  System.out.println(s1==s3);         //false(because s3 refers to instance created in
nonpool)
 }
}
```

Output:

```
true
false
```

---

3) String compare by compareTo() method

The above code, demonstrates the use of == operator used for comparing two Stringobjects.

---

4) By Using compareTo() method

The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greaterthan second string.

Suppose s1 and s2 are two String objects. If:s1 ==

s2 : The method returns 0.

s1 > s2 : The method returns a positive value.s1 <

s2 : The method returns a negative value.

```
class Teststringcomparison4{
 public static void main(String args[]){
  String s1="Sachin";
  String s2="Sachin";String
  s3="Ratan";
  System.out.println(s1.compareTo(s2));//0
  System.out.println(s1.compareTo(s3));//1(because s1>s3)
  System.out.println(s3.compareTo(s1));//-1(because  s3  <  s1  )
 }
}
```

Output:

```
0
1
-1
```

ARRAY

= *Array is collection of similar datatype*
= object is storer in heap memory .

Advantages of array :

= Code Optimization: It makes the code optimized, we can retrieve or sort the dataefficiently.

=Random access: We can get any data located at an index position.
= single name similar data is stores .

Disadvantages of array :

= different datatype cannot be used .
= size of array is fixed .
= memory wastage .
= reduce performance .

Example of Java Array
Let's see the simple example of java array, where we are going to declare, instantiate,initialize and traverse an array.

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];          //declaration  and  instantiation
a[0]=10;                      //initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
 //traversing array
for(int i=0;i<a.length;i++)                    //length is the property of array
System.out.println(a[i]);
}}
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array
We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};            //declaration,  instantiation  and  initializationLet's
```

see the simple example to print this array.

```
//Java Program to illustrate the use of declaration, instantiation
```

```
//and initialization of Java array in a single line

class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};            //declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)           //length is the property of array
System.out.println(a[i]);
}
}
```

Output:

```
33
3
4
5
```

For-each Loop for Java Array

We can also print the Java array using for-each loop
. The Java for-each loop prints the array elements one by one. It holds an arrayelement in a variable, then executes the body of the loop.
The syntax of the for-each loop is given below:

```
for(data_type  variable:array){
//body of the loop
}
```

Let us see the example of print the elements of Java array using the for-each loop.

```
                         //Java Program to print the array elements using for-each loop
class Testarray1{
public static void main(String args[]){int
arr[]={33,3,4,5};
                      //printing array using for-each loop
for(int i:arr)
System.out.println(i);
}
}
```

Output:

```
33
3
4
5
```

Passing Array to a Method in Java

We can pass the java array to method so that we can reuse the same logic on anyarray.
Let's see the simple example to get the minimum number of an array using a method.

```
//Java Program to demonstrate the way of passing an array
//to method. class
Testarray2{
//creating a method which receives an array as a parameterstatic void
min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
  min=arr[i];

System.out.println(min);
}

public static void main(String args[]){
int a[]={33,3,4,5} ;          //declaring and initializing an array
min(a);                        //passing array to method
}
}
```

Collection:-

*The **Collection in Java** is a framework that provides an architecture to store andmanipulate the group of objects.*
Java Collections can achieve all the operations that you perform on a data such assearching, sorting, insertion, manipulation, and deletion.
Java Collection means a single unit of objects. Java Collection framework providesmany interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).
*The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main "root" interfaces of Java collection classes.*

***java.util.Collection**:- The java. util. Collections package is **the package that containsthe Collections class**. Collections class is basically used with the static methods that operate on the collections or return the collection. The best general purpose or 'primary' implementations are likely **ArrayList , HashMap , and HashSet** .*
***HashMap :-** Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc.*

Points to remember:-

- *Java HashMap contains values based on the key.*
- *Java HashMap contains only unique keys.*
- *Java HashMap may have one null key and multiple null values.*
- *Java HashMap is non synchronized.*

- *Java HashMap maintains no order.*

1. clear()	*It is used to remove all of the mappings from this map.*
2. isEmpty()	*It is used to return true if this map contains no key-value mappings*
3. Object clone()	*It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.*
4. entrySet()	*It is used to return a collection view of themappings contained in this map.*
5. keySet()	*It is used to return a set view of the keys contained in this map.*
6. put(Object key, Objectvalue)	*It is used to insert an entry in the map.*
7. remove(Object key)	*It is used to delete an entry for the specified key.*
8. containsKey(Object key)	*This method returns true if some key equal to thekey exists within the map, else return false*
9. replace(K key, V value)	*It replaces the specified value for a specified key.*
10. size()	*This method returns the number of entries in themap.*

***Java.util.Map**:- Java HashSet class is used to create a collection that uses a hash table forstorage. It inherits the AbstractSet class and implements Set interface.*
The important points about Java HashSet class are:

- *HashSet stores the elements by using a mechanism called **hashing.***
- *HashSet contains unique elements only.*
- *HashSet allows null value.*
- *HashSet class is non synchronized.*
- *HashSet doesn't maintain the insertion order. Here, elements are inserted on thebasis of their hashcode.*

Methods of HashSet:-

Method	*Description*
add(E e)	*It is used to add the specified element to this set if it is notalready present.*
clear()	*It is used to remove all of the elements from the set.*
clone()	*It is used to return a shallow copy of this HashSet instance:the elements themselves are not cloned.*

contains(Object o)	It is used to return true if this set contains the specified element.
isEmpty()	It is used to return true if this set contains no elements.
iterator()	It is used to return an iterator over the elements in this set.
remove(Object o)	It is used to remove the specified element from this set if it ispresent.
size()	It is used to return the number of elements in the set.
spliterator()	It is used to create a late-binding and fail-fast Spliterator overthe elements in the set.

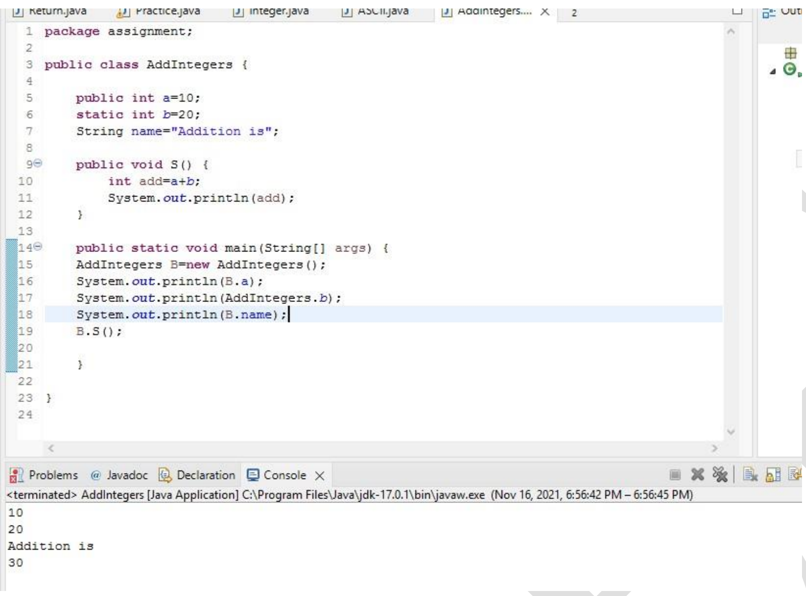
Pattern Program:-

1. Star Pattern:- By using increment:
```
Package demo;
Public class Star {

Public static void main(string[]args){for(int
a=1;a<=5;a++){
for(int b=1;b<=a;b++){
system.out.println("*);
}
system.out.println();
}
}
}
```

2. Star Pattern:- By using decrement:
```
Package demo;
Public class Star {
Public static void main(string[]args){for(int
a=1;a<=5;a++){
for(int b=5;b>=a;b--){
system.out.println("*");
}
system.out.println();
}
}
}
```

Logical Programs:-

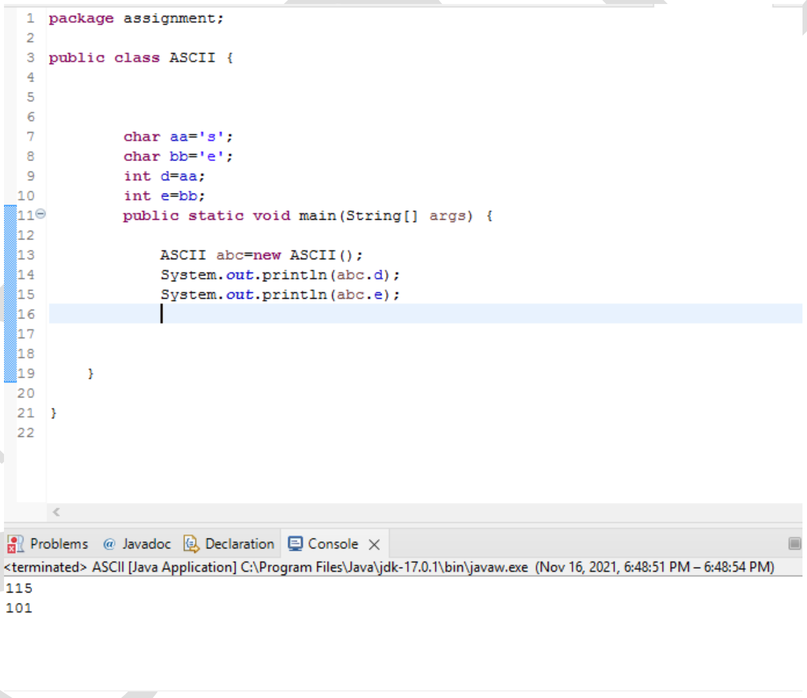
1. ***Adding an Integers:-***



```
package assignment;

public class AddIntegers {

    public int a=10;
    static int b=20;
    String name="Addition is";

    public void S() {
        int add=a+b;
        System.out.println(add);
    }

    public static void main(String[] args) {
    AddIntegers B=new AddIntegers();
    System.out.println(B.a);
    System.out.println(AddIntegers.b);
    System.out.println(B.name);
    B.S();

    }

}
```

```
<terminated> AddIntegers [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 16, 2021, 6:56:42 PM – 6:56:45 PM)
10
20
Addition is
30
```

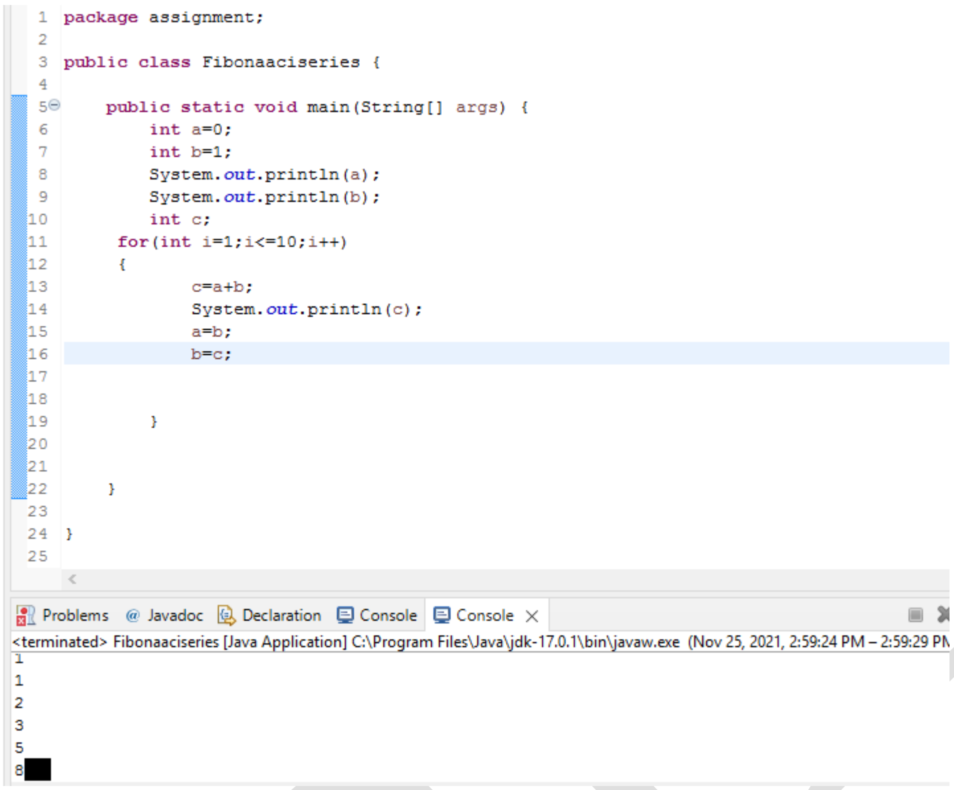
2. ***ASCII Pattern:***



```
package assignment;

public class ASCII {



        char aa='s';
        char bb='e';
        int d=aa;
        int e=bb;
        public static void main(String[] args) {

            ASCII abc=new ASCII();
            System.out.println(abc.d);
            System.out.println(abc.e);



    }

}
```

```
<terminated> ASCII [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 16, 2021, 6:48:51 PM – 6:48:54 PM)
115
101
```

3. *Factorial of number:*

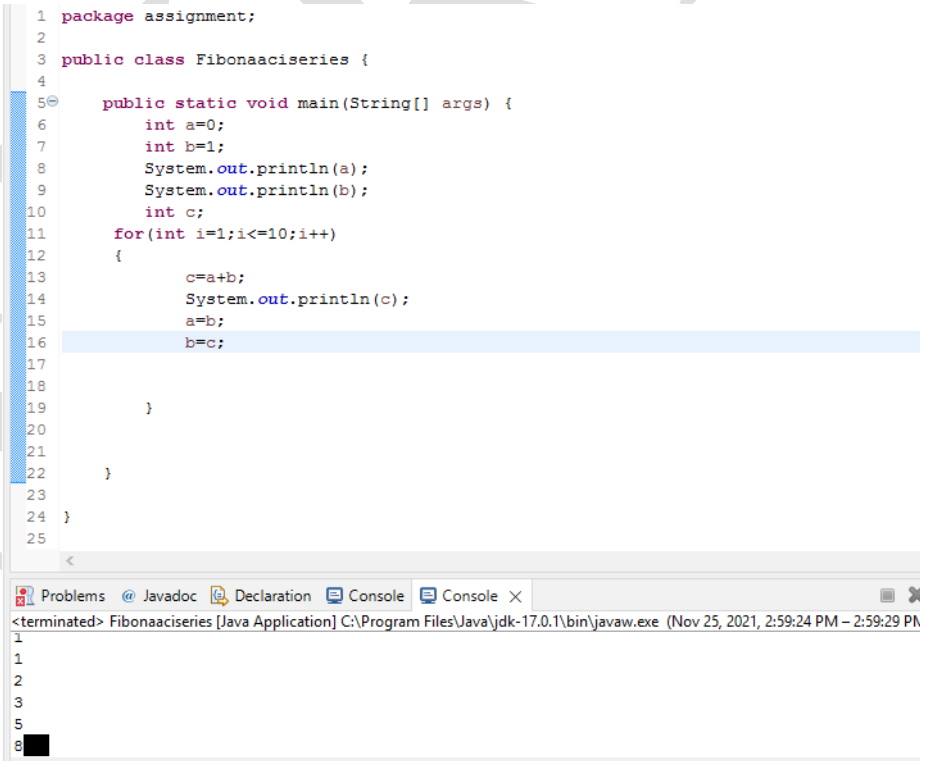


```
package assignment;

public class Fibonaaciseries {

    public static void main(String[] args) {
        int a=0;
        int b=1;
        System.out.println(a);
        System.out.println(b);
        int c;
     for(int i=1;i<=10;i++)
     {
           c=a+b;
           System.out.println(c);
           a=b;
           b=c;


        }


    }

}
```

Problems Javadoc Declaration Console Console

```
<terminated> Fibonaaciseries [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 25, 2021, 2:59:24 PM – 2:59:29 PM
1
1
2
3
5
8
```

4. *Fibonaaci Series:*

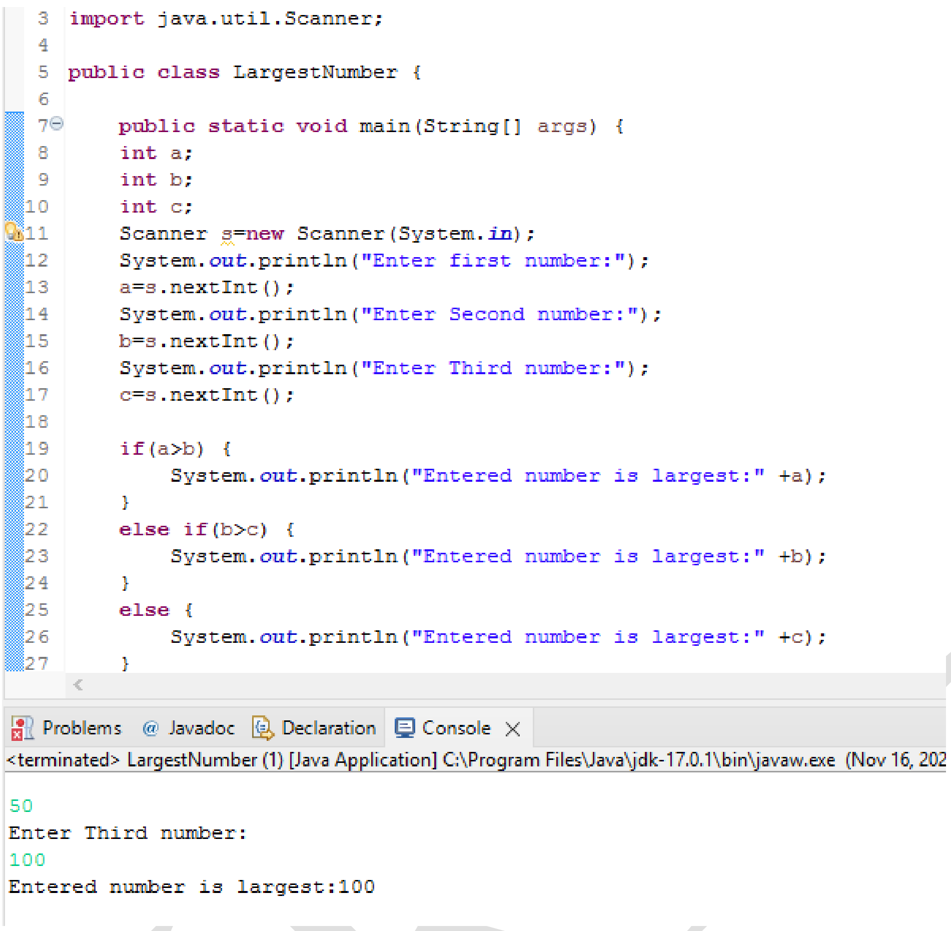


```
package assignment;

public class Fibonaaciseries {

    public static void main(String[] args) {
        int a=0;
        int b=1;
        System.out.println(a);
        System.out.println(b);
        int c;
     for(int i=1;i<=10;i++)
     {
           c=a+b;
           System.out.println(c);
           a=b;
           b=c;


        }


    }

}
```

Problems Javadoc Declaration Console Console

```
<terminated> Fibonaaciseries [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 25, 2021, 2:59:24 PM – 2:59:29 PM
1
1
2
3
5
8
```

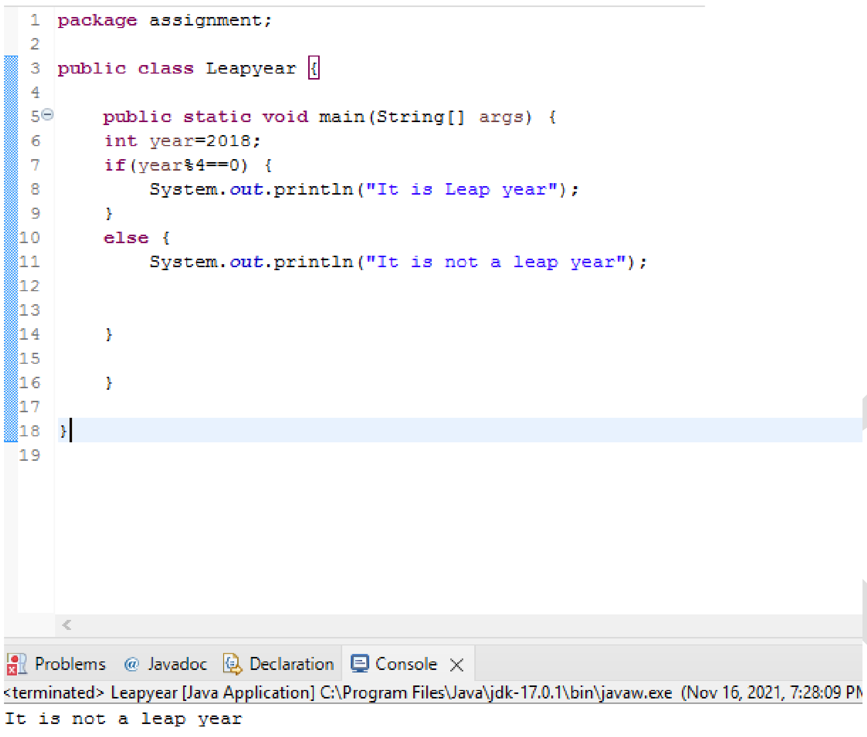
5. *Largest Number:*



```
import java.util.Scanner;

public class LargestNumber {

    public static void main(String[] args) {
    int a;
    int b;
    int c;
    Scanner s=new Scanner(System.in);
    System.out.println("Enter first number:");
    a=s.nextInt();
    System.out.println("Enter Second number:");
    b=s.nextInt();
    System.out.println("Enter Third number:");
    c=s.nextInt();

    if(a>b) {
        System.out.println("Entered number is largest:" +a);
    }
    else if(b>c) {
        System.out.println("Entered number is largest:" +b);
    }
    else {
        System.out.println("Entered number is largest:" +c);
    }
```

```
<terminated> LargestNumber (1) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (Nov 16, 202

50
Enter Third number:
100
Entered number is largest:100
```

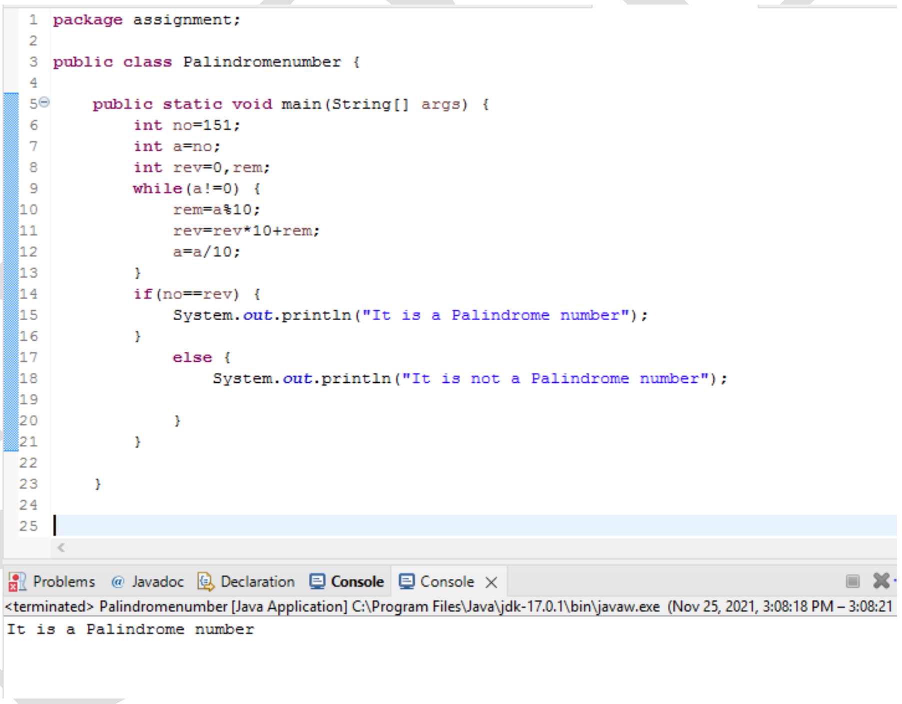
6. *Leap Year:*



```
package assignment;

public class Leapyear {

    public static void main(String[] args) {
    int year=2018;
    if(year%4==0) {
        System.out.println("It is Leap year");
    }
    else {
        System.out.println("It is not a leap year");


    }

    }

}

```

```
<terminated> Leapyear [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 16, 2021, 7:28:09 PM
It is not a leap year
```

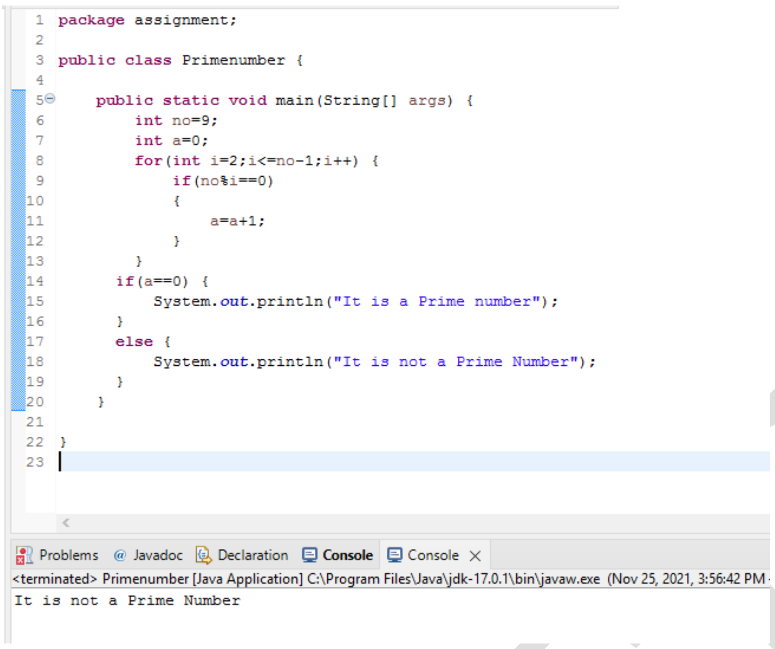
7. ***Palindrome number:***



```
package assignment;

public class Palindromenumber {

    public static void main(String[] args) {
        int no=151;
        int a=no;
        int rev=0,rem;
        while(a!=0) {
            rem=a%10;
            rev=rev*10+rem;
            a=a/10;
        }
        if(no==rev) {
            System.out.println("It is a Palindrome number");
        }
            else {
                System.out.println("It is not a Palindrome number");

            }
        }

    }

```

```
<terminated> Palindromenumber [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 25, 2021, 3:08:18 PM – 3:08:21
It is a Palindrome number
```

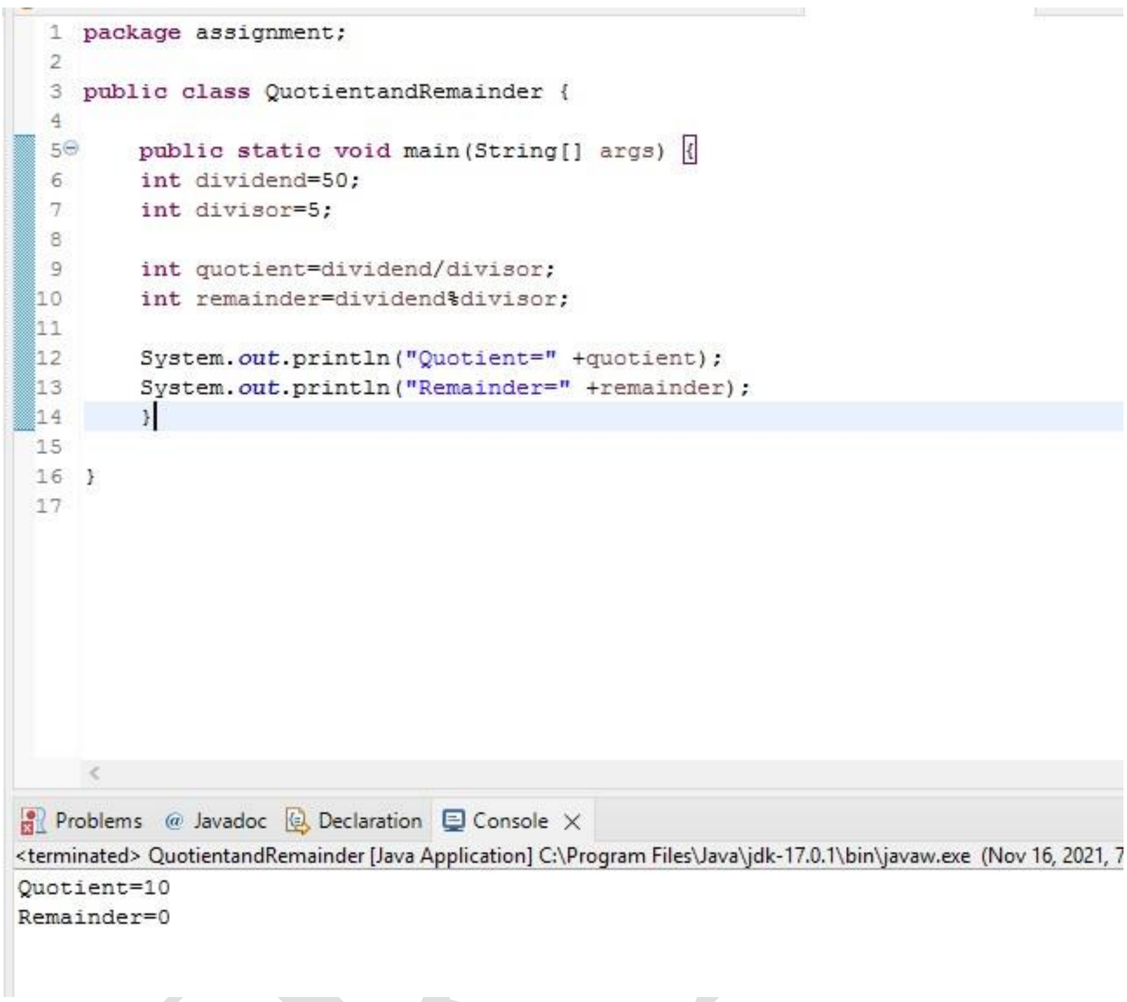
8. ***Prime number:***



```java
package assignment;

public class Primenumber {

    public static void main(String[] args) {
        int no=9;
        int a=0;
        for(int i=2;i<=no-1;i++) {
            if(no%i==0)
            {
                a=a+1;
            }
        }
      if(a==0) {
          System.out.println("It is a Prime number");
      }
      else {
          System.out.println("It is not a Prime Number");
      }
    }

}
```

```
<terminated> Primenumber [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 25, 2021, 3:56:42 PM
It is not a Prime Number
```

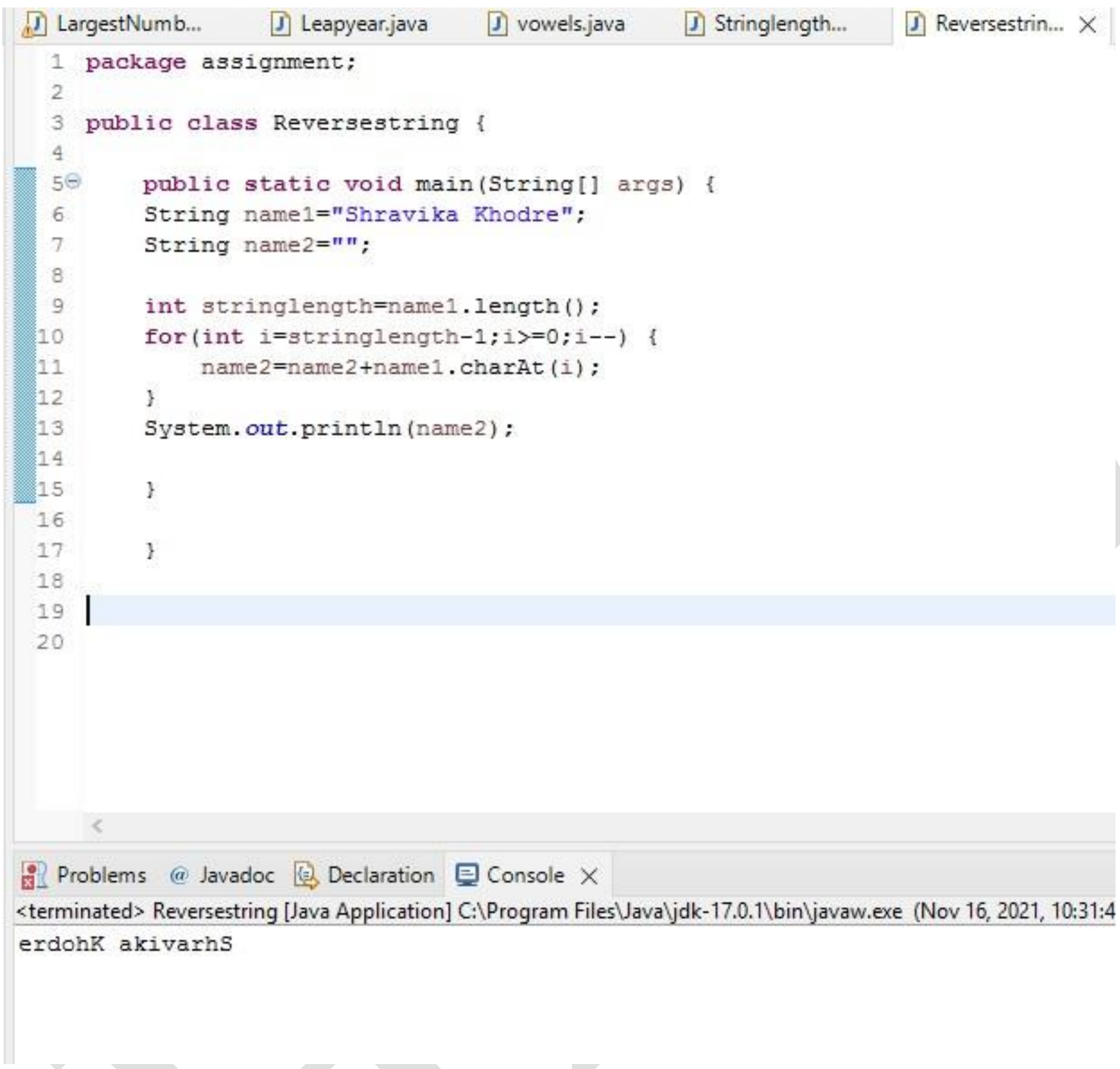
9. *Quotient and remainder:*



```
package assignment;

public class QuotientandRemainder {

    public static void main(String[] args) {
    int dividend=50;
    int divisor=5;

    int quotient=dividend/divisor;
    int remainder=dividend%divisor;

    System.out.println("Quotient=" +quotient);
    System.out.println("Remainder=" +remainder);
    }

}
```

```
<terminated> QuotientandRemainder [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 16, 2021, 7
Quotient=10
Remainder=0
```

10. Reverse String:



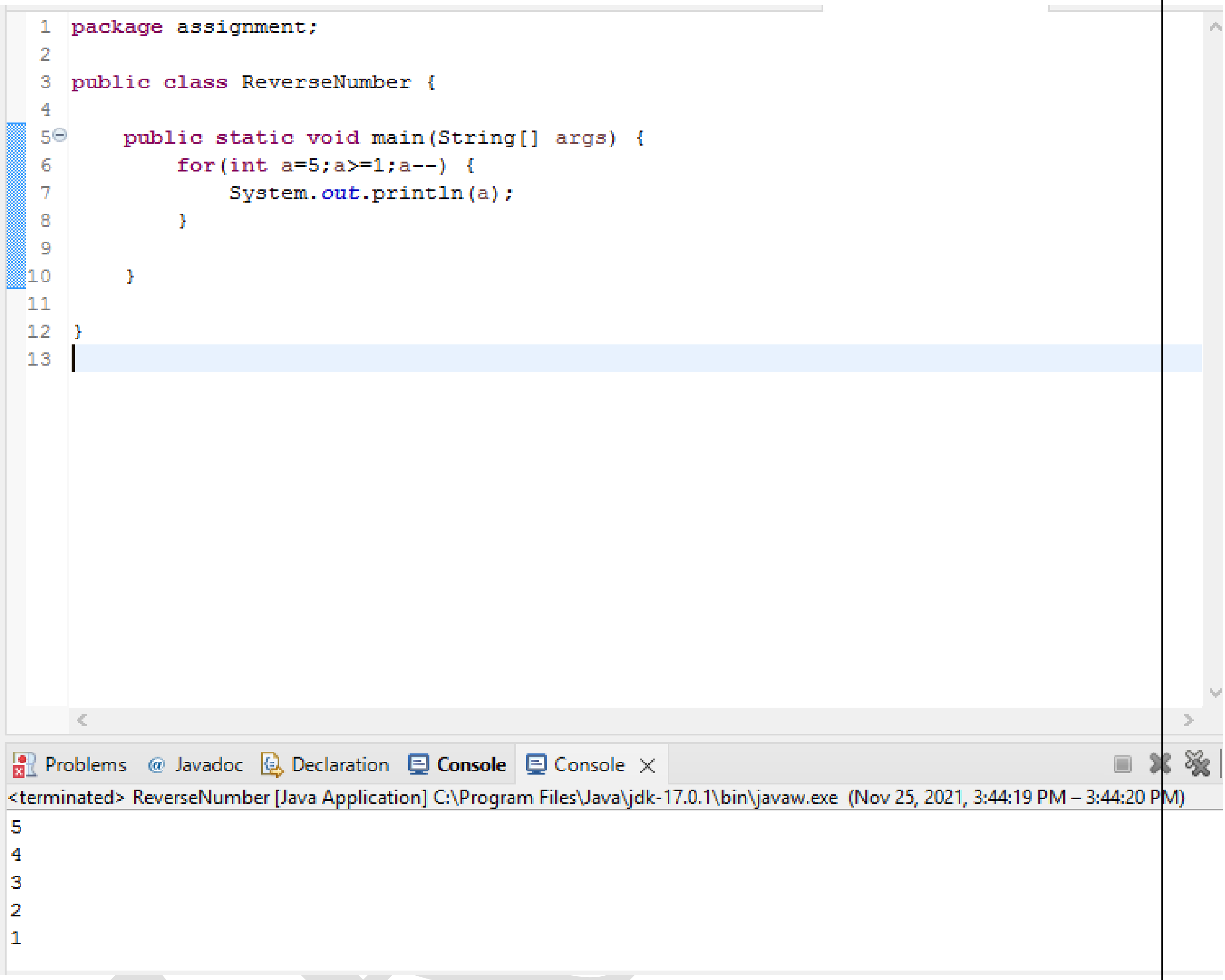
LargestNumb... Leapyear.java vowels.java Stringlength... Reversestrin...

```java
package assignment;

public class Reversestring {

    public static void main(String[] args) {
    String name1="Shravika Khodre";
    String name2="";

    int stringlength=name1.length();
    for(int i=stringlength-1;i>=0;i--) {
        name2=name2+name1.charAt(i);
    }
    System.out.println(name2);

    }

    }


```

Problems Javadoc Declaration Console

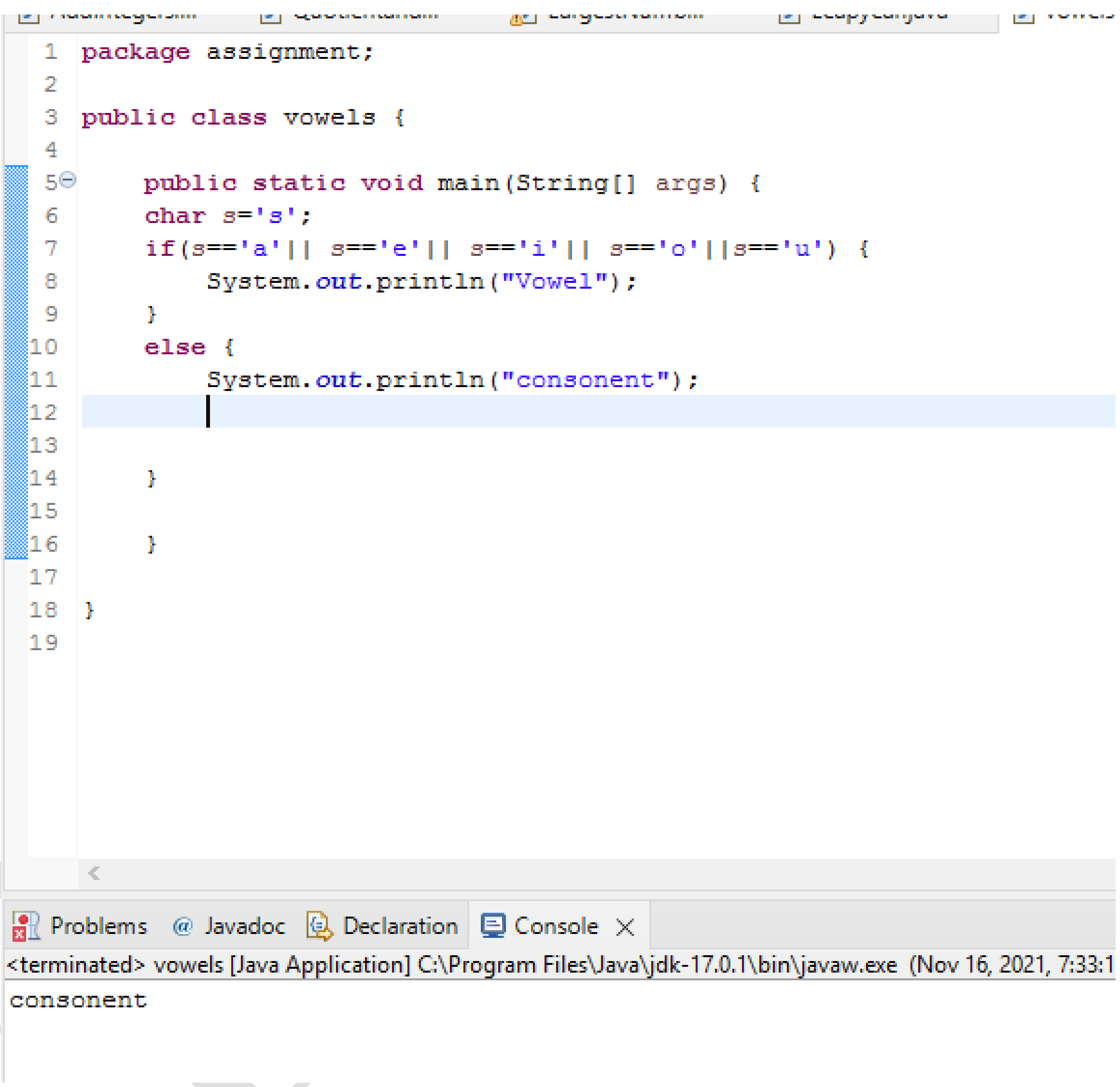
```
<terminated> Reversestring [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 16, 2021, 10:31:4
erdohK akivarhS
```

11. Reverse Number:


```java
package assignment;

public class ReverseNumber {

	public static void main(String[] args) {
		for(int a=5;a>=1;a--) {
			System.out.println(a);
		}

	}

}
```

```
<terminated> ReverseNumber [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (Nov 25, 2021, 3:44:19 PM – 3:44:20 PM)
5
4
3
2
1
```

12. Vowels and Consonant:



```
package assignment;

public class vowels {

    public static void main(String[] args) {
    char s='s';
    if(s=='a'|| s=='e'|| s=='i'|| s=='o'||s=='u') {
        System.out.println("Vowel");
    }
    else {
        System.out.println("consonent");


    }

    }

}
```

```
<terminated> vowels [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (Nov 16, 2021, 7:33:1
consonent
```