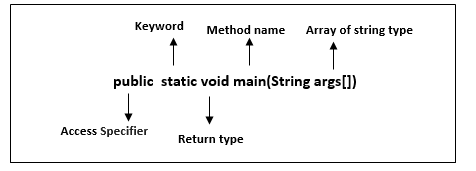
**1. Explain Java Main Method public static void main (String[] args)**

**Main():** It is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program. The syntax of the main() method is:



**public:** It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to JVM.

**static:** You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.

**void:** In Java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.

**2.What is java?**

Java is a **programming language** and a **Platform Independent**. Java is a high level, robust, object-oriented, and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the 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 Independent**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform Independent.

**Java Example**

Let's have a quick look at Java programming example. A detailed description of Hello Java example is available in next sections.

**Simple.java**

public class Simple

{

public static void main(String args[])

{

System.out.println("Hello Java");

}

}

**Application**

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

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 Java programming:

**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 a web application. Currently, Servlet, JSP, 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, EJB 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.

**3. Mention some features of Java?**



**1. Simple**

Java is one of the simplest programming languages to learn and master in order to grasp the concept of Object-oriented learning.

For experienced developers, there was once a problem of unreferenced objects clogging up the memory. With Java, that problem is now solved as Java was one of the first programming languages to bring in the concept of Automatic Garbage Collection. When Java came into being, a lot of developers were already working in the then programming language C++. In order to tackle the migration problem, the Java syntax is actually very similar to the C++ syntax which made it easier for developers to migrate from C++ to Java.

**2. Object-Oriented Programming Language**

This is one of the primary reasons why Java is so popular amongst developers. Java strictly conforms to the rules of Object-Oriented Programming or OOP. Object-oriented development includes concepts of Objects and Classes and many more. This enables developers to have a wide variety of options for designing their software.

Java also supports

• Polymorphism

• Inheritance

• Abstraction

• Encapsulation

**3. Platform-Independent**

Being platform-independent is important for a programming language as the developer circles range from windows users to Linux users. Java does that by having the concept of a Java Virtual Machine. **It is also known as a write-once run anywhere (WORA).** This is where it beats the other programming languages by a huge mark. Previously all programming languages generated machine-level code for the machine environment. Hence it was very difficult for developers to collaborate if they were using different environments to code the same software. This is where Java came in with a software-oriented platform that had 2 components namely an API and a Runtime Environment.

If we break the discussion to the basics what Java does is as soon as the program compiles, the compiler generates bytecode for that specific program.

Different operating systems such as MAC or Linux can then run the bytecode which makes it very flexible.

**4. Secure Language**

Security is everybody’s prime concern in today’s world. Java promises security by implementing some special features like It completely obliterates(destroy) the use of explicit pointers.

Java, unlike other languages, runs all its programs inside a sandbox of its own called the Java Virtual Machine so that any errors or crashes do not harm the external operating system, thus making it secure and efficient at the same time. The Class Loader dynamically allocates the classes defined in the program to the Java Runtime environment by separating the classes that are local to the machine and those that are imported from other network sources.

Sometimes malicious code can try to access objects outside their permission domains. The bytecode verifier prevents this by obliterating such code.

Java can determine what resources a particular class can access such as reading from a disk and writing data to memory. It does so with the help of a security manager.

**4. Is Java 100% Object Oriented Language?**

Java supports property 1, 2, 3, 4 and 6 but fails to support property 5 and 7 given above. Java language is not a Pure Object Oriented Language as it contain these properties:

* **Primitive Data Type ex. int, long, bool, float, char, etc. as Objects:** Smalltalk is a “pure” object-oriented programming language unlike Java and C++ as there is no difference between values which are objects and values which are primitive types. In Smalltalk, primitive values such as integers, Booleans and characters are also objects.  
  In Java, we have predefined types as non-objects (primitive types).
* int a = 5;
* System.out.print(a);
* **The static keyword:**When we declares a class as static then it can be used without the use of an object in Java. If we are using static function or static variable then we can’t call that function or variable by using dot(.) or class object defying object oriented feature.
* **Wrapper Class:** Wrapper class provides the mechanism to convert primitive into object and object into primitive. In Java, you can use Integer, Float etc. instead of int, float etc. We can communicate with objects without calling their methods. ex. using arithmetic operators.
* String s1 = "ABC" + "A" ;

Even using Wrapper classes does not make Java a pure OOP language, as internally it will use the operations like Unboxing and Auto boxing. So if you create instead of int Integer and do any mathematical operation on it, under the hoods Java is going to use primitive type int only.

|  |
| --- |
| public class BoxingExample  {  public static void main(String[] args)  {  Integer i = new Integer(10);  Integer j = new Integer(20);  Integer k = new Integer(i.intValue() + j.intValue());  System.out.println("Output: "+ k);  }  } |

**In the above code, there are 2 problems where Java fails to work as pure OOP:**

* 1. While creating Integer class you are using primitive type “int” i.e. numbers 10, 20.
  2. While doing addition Java is using primitive type “int”.

**5. What is the difference between Object-oriented programming language and Object-based programming language?**

**Object-Oriented Languages (OOP)** follow all the concepts of OOPs whereas, **Object-based languages** don't follow all the concepts of OOPs like inheritance and polymorphism.

Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object.

**6.What is the difference between Declaration and Definition**

**in Java?**

| **Sr. No.** | **Key** | **Declaration** | **Definition** |
| --- | --- | --- | --- |
| 1 | Concept | The concept of declaration includes informing the compiler about properties of the variable such as its name, type of value it holds and the initial value if any it takes.  int a; // declaration of variable | While the definition is basically the actual implementation and memory location of function and about memory for the variable is allocated during the definition of the variable.  a=10; // definition of variable |
| 2 | Exception in C | Both declaration and definition take place at the same time in the case of c language. | In other languages such as Java both occur at different places. |
| 3 | Number of occurrences | The declaration could be done multiple times either of a variable or of function. | Variable or function could be defined only once. |
| 4 | Memory allocation | Memory has not been allocated during the declaration of a variable or function. | Memory has been allocated during the definition of a variable or function. |

**7,8,9: What is JRE,JVM,JDK and why is it required?**

Following are the important differences between JDK,JRE and JVM

| **Sr. No.** | **Key** | **JDK** | **JRE** | **JVM** |
| --- | --- | --- | --- | --- |
| 1 | Definition | JDK (Java Development Kit) is a software development kit to develop applications in Java. In addition to JRE, JDK also contains number of development tools (compilers, Javadoc, Java Debugger etc.). | JRE (Java Runtime Environment) is the implementation of JVM and is defined as a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run applications written in Java programming. | JVM (Java Virtual Machine) is an abstract machine that is platform-dependent and has three notions(Idea or Conception) as a specification, a document that describes requirement of JVM implementation, implementation, a computer program that meets JVM requirements, and instance, an implementation that executes Java byte code provides a runtime environment for executing Java byte code. |
| 2 | Prime functionality | JDK is primarily used for code execution and has prime functionality of development. | On other hand JRE is majorly responsible for creating environment for code execution. | JVM on other hand specifies all the implementations and responsible to provide these implementations to JRE. |
| 3 | Platform Independence | JDK is platform dependent i.e. for different platforms different JDK required. | Like of JRE is also platform dependent. | JVM is platform independent. |
| 4 | Tools | As JDK is responsible for prime development so it contains tools for developing, debugging and monitoring java application. | On other hand JRE does not contain tools such as compiler or debugger etc. Rather it contains class libraries and other supporting files that JVM requires to run the program. | JVM does not include software development tools. |
| 5 | Implementation | JDK = Java Runtime Environment (JRE) + Development tools | JRE = Java Virtual Machine (JVM) + Libraries to run the application | JVM = Only Runtime environment for executing the Java byte code. |

**Why is required?**

**JRE:**- JRE (Java Runtime Environment) is an installation package that **provides an environment to only run(not develop) the java program(or application)onto your machine**. JRE is only used by those who only want to run Java programs that are end-users of your system.

**JDK: -** The JDK is a development environment for building applications, applets, and components using the Java programming language. The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

**JVM:-** JVM is specifically **responsible for converting bytecode to machine-specific code** and is necessary in both JDK and JRE. It is also platform-dependent and performs many functions, including memory management and security.

**10.What is an Object in Java?**

An entity that has state and behaviour is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

* **State:** represents the data (value) of an object.
* **Behaviour:** represents the behaviour (functionality) of an object such as deposit, withdraw, etc.
* **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

**11.What is a Class in Java?**

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* Fields
* Methods
* Constructors
* Blocks
* Nested class and interface

**12.What is Constructor in Java?**

In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

**Note:** It is called constructor because it constructs the values at the time of object creation. 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.

**constructor Rules for creating Java**

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized.

## Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor):-

A constructor is called "Default Constructor" when it doesn't have any parameter.

### Syntax of default constructor:

1. <class\_name>(){}

## Example of default constructor

|  |
| --- |
| In this example, we are creating the no-arg constructor in the Bike class. It will be invoked(आवाहन करना ) at the time of object creation. |

1. //Java Program to create and call a default constructor
2. **class** Bike1{
3. //creating a default constructor
4. Bike1(){System.out.println("Bike is created");}
5. //main method
6. **public** **static** **void** main(String args[]){
7. //calling a default constructor
8. Bike1 b=**new** Bike1();
9. }
10. }
11. **Parameterized constructor:-**

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

1. //Java Program to demonstrate the use of the parameterized constructor.
2. **class** Student4{
3. **int** id;
4. String name;
5. //creating a parameterized constructor
6. Student4(**int** i, String n){
7. id = i;
8. name = n;
9. }
10. //method to display the values
11. **void** display(){System.out.println(id+" "+name);}
12. **public** **static** **void** main(String args[]){
13. //creating objects and passing values
14. Student4 s1 = **new** Student4(111,"Karan");
15. Student4 s2 = **new** Student4(222,"Aryan");
16. //calling method to display the values of object
17. s1.display();
18. s2.display();
19. }
20. }

**13.What is Local Variable and Instance Variable?**

* **Instance variables** − Instance variables are declared in a class, but outside a method. When space is allocated for an object in the heap, a slot for each instance variable value is created. Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
* **Local variables** − Local variables are declared in methods, constructors, or blocks. Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.

|  |  |
| --- | --- |
| Instance Variable | Local Variable |
| They are defined in class but outside the body of methods. | They are defined as a type of variable declared within programming blocks or subroutines. |
| These variables are created when an object is instantiated and are accessible to all constructors, methods, or blocks in class. | These variables are created when a block, method or constructor is started and the variable will be destroyed once it exits the block, method, or constructor. |
| These variables are destroyed when the object is destroyed. | These variables are destroyed when the constructor or method is exited. |
| It can be accessed throughout the class. | Its access is limited to the method in which it is declared. |
| They are used to reserving memory for data that the class needs and that too for the lifetime of the object. | They are used to decreasing dependencies between components I.e., the complexity of code is decreased. |
| These variables are given a default value if it is not assigned by code. | These variables do not always have some value, so there must be a value assigned by code. |

**14.What are the OOPs concepts?**

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

[**Object**](https://www.javatpoint.com/object-and-class-in-java)

* Any entity that has state and behaviour is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
* An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.
* **Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviours like wagging the tail, barking, eating, etc.

**Class**

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

**Inheritance**

When one object acquires all the properties and behaviours of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**

If one task is performed in differentways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

#### Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

### Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**15.What is Inheritance in Java?**

Inheritance is the procedure in which one class inherits the attributes and methods of another class.

**16.What is Polymorphism?**

**Polymorphism in Java** is a concept by which we can perform *a*single action in different ways*.* Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

**17.What are the types of Polymorphism?**

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

**18.What is Method Overloading ?**

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading.**

1. **class** Adder{
2. **static** **int** add(**int** a,**int** b){**return** a+b;}
3. **static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}
4. }
5. **class** TestOverloading1{
6. **public** **static** **void** main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(11,11,11));
9. }}

**19.What is Method Overriding?**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

#### Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

//Java Program to illustrate the use of Java Method Overriding

1. //Creating a parent class.
2. **class** Vehicle{
3. //defining a method
4. **void** run(){System.out.println("Vehicle is running");}
5. }
6. //Creating a child class
7. **class** Bike2 **extends** Vehicle{
8. //defining the same method as in the parent class
9. **void** run(){System.out.println("Bike is running safely");}
11. **public** **static** **void** main(String args[]){
12. Bike2 obj = **new** Bike2();//creating object
13. obj.run();//calling method
14. }
15. }

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

**20.What is Abstraction in Java?**

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

**21.What is Abstract Class in Java?**

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

#### Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors and static methods also.](https://www.javatpoint.com/java-constructor)
* It can have final methods which will force the subclass not to change the body of the method.



**Example of abstract class**

1. **abstract** **class** A{}

**22.What is Abstract Method?**

A method which is declared as abstract and does not have implementation is known as an abstract method.

**Example of abstract method**

**abstract** **void** printStatus()**;** //no method body and abstract

**23.What is Interface in Java?**

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [abstraction.](https://www.javatpoint.com/abstract-class-in-java)

There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java)

Java Interface also **represents the IS-A relationship**.

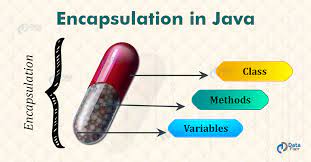
1. //Interface declaration: by first user
2. **interface** Drawable{
3. **void** draw();
4. }
5. //Implementation: by second user
6. **class** Rectangle **implements** Drawable{
7. **public** **void** draw(){System.out.println("drawing rectangle");}
8. }
9. **class** Circle **implements** Drawable{
10. **public** **void** draw(){System.out.println("drawing circle");}
11. }
12. //Using interface: by third user
13. **class** TestInterface1{
14. **public** **static** **void** main(String args[]){
15. Drawable d=**new** Circle();//In real scenario, object is provided by method e.g. getDrawable()
16. d.draw();
17. }}

Output:

drawing circle

**24.What is Encapsulation in Java?**

**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.

The **Java Bean** class is the example of a fully encapsulated class.

1. //A Java class which is a fully encapsulated class.
2. //It has a private data member and getter and setter methods.
3. **package** com.javatpoint;
4. **public** **class** Student{
5. //private data member
6. **private** String name;
7. //getter method for name
8. **public** String getName(){
9. **return** name;
10. }
11. //setter method for name
12. **public** **void** setName(String name){
13. **this**.name=name
14. }
15. }

**25.What is String in Java?**

In java string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java)

of characters works same as Java string. For example:

**char**[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=**new** String(ch);

is same as: String s="javatpoint";

**26.Why are strings immutable in Java?**

A String is an unavoidable(mandatory) 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 means unmodifiable or unchangeable. Once String object is created its data or state can't be changed but a new String object is created.

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

1. **class** Testimmutablestring{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

**27.What is the difference between equals(=) method and double equal operator (==) in Java?**

Both equals() method and the == operator are used to compare two objects in Java. == is an operator and equals() is method. But == operator compares reference or memory location of objects in a heap, whether they point to the same location or not.

**28.How to convert Integer to String in Java?**

# Java Convert int to String

We can convert **int to String in java** using **String.valueOf()** **and Integer.toString()** methods. Alternatively, we can **use String.format()** method, string concatenation operator etc.

## 1) String.valueOf()

The String.valueOf() method converts int to String. The valueOf() is the static method of String class. The **signature** of valueOf() method is given below:

**public** **static** String valueOf(**int** i)

## Java int to String Example using String.valueOf()

Let's see the simple code to convert int to String in java.

1. **int** i=10;
2. String s=String.valueOf(i);//Now it will return "10"

Let's see the simple example of converting String to int in java.

**public** **class** IntToStringExample1{

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

1. **int** i=200;
2. String s=String.valueOf(i);
3. System.out.println(i+100);//300 because + is binary plus operator
4. System.out.println(s+100);//200100 because + is string concatenation operator  }}

Output:

300

200100

**2) Integer.toString()**

The Integer.toString() method converts int to String. The toString() is the static method of Integer class. The **signature** of toString() method is given below:

**public** **static** String toString(**int** i)

## Java int to String Example using Integer.toString()

Let's see the simple code to convert int to String in java using Integer.toString() method.

**int** i=10;

String s=Integer.toString(i);//Now it will return "10"

Let's see the simple example of converting String to int in java.

**public** **class** IntToStringExample2{

1. **public** **static** **void** main(String args[]){
2. **int** i=200;
3. String s=Integer.toString(i);
4. System.out.println(i+100);//300 because + is binary plus operator
5. System.out.println(s+100);//200100 because + is string concatenation operator
6. }}
7. Output:
8. 300
9. 200100

## 3) String.format()

The String.format() method is used to format given arguments into String. It is introduced since Jdk 1.5.

**public** **static** String format(String format, Object... args)

**Java int to String Example using String.format()**

Let's see the simple code to convert int to String in java using String.format() method.

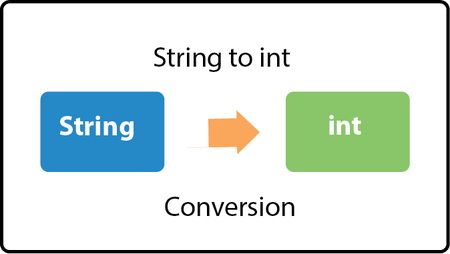
1. **public** **class** IntToStringExample3{
2. **public** **static** **void** main(String args[]){
3. **int** i=200;
4. String s=String.format("%d",i);
5. System.out.println(s);
6. }}

***29.How to convert String to Integer in Java?***

We can convert **String to an int in java** using [Integer.parseInt () method. To convert](https://www.javatpoint.com/java-integer-parseint-method)[String](https://www.javatpoint.com/java-string)

[, we can use](https://www.javatpoint.com/java-integer-parseint-method)[Integer.valueOf()](https://www.javatpoint.com/java-integer-valueof-method)

method which returns instance of Integer class.



## Scenario

It is generally used if we have to perform mathematical operations on the string which contains a number. Whenever we receive data from Text Field or Text Area, entered data is received as a string. If entered data is in number format, we need to convert the string to an int. To do so, we use Integer.parseInt() method.

## Signature

The parseInt() is the static method of Integer class. The **signature** of parseInt() method is given below:

**public** **static** **int** parseInt(String s)

## Java String to int Example: Integer.parseInt()

Let's see the simple code to convert a string to an int in java.

1. **int** i=Integer.parseInt("200");

Let's see the simple example of converting String to int in Java.

//Java Program to demonstrate the conversion of String into int

//using Integer.parseInt() method

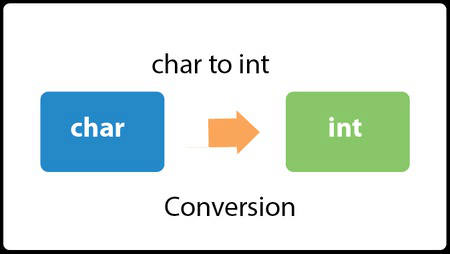
1. **public** **class** StringToIntExample1{
2. **public** **static** **void** main(String args[]){
3. //Declaring String variable
4. String s="200";
5. //Converting String into int using Integer.parseInt()
6. **int** i=Integer.parseInt(s);
7. //Printing value of i
8. System.out.println(i);
9. }}

Output:

1. 200

**30.How to convert Char to Integer in Java?**

We can convert **char to int in java** using various ways. If we direct assign char variable to int, it will return ASCII value of given character.



If char variable contains int value, we can get the int value by calling *Character.getNumericValue(char)* method. Alternatively, we can use String.valueOf(char) method.

## 1) Java char to int Example: Get ASCII value

Let's see the simple code to convert char to int in java.

**public** **class** CharToIntExample1{

1. **public** **static** **void** main(String args[]){
2. **char** c='a';
3. **char** c2='1';
4. **int** a=c;
5. **int** b=c2;
6. System.out.println(a);
7. System.out.println(b);
8. }}

OUTPUT97

97

49

## 2) Java char to int Example: Character.getNumericValue()

Let's see the simple code to convert char to int in java using Character.getNumericValue(char) method which returns an integer value.

**public** **class** CharToIntExample2{

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

1. **char** c='1';
2. **int** a=Character.getNumericValue(c);
3. System.out.println(a);
4. }}

OUTPUT

1

## 3) Java char to int Example: String.valueOf()

Let's see another example which returns integer value of specified char value using String.valueOf(char) method

public class CharToIntExample3{

1. public static void main(String args[]){
2. char c='1';
3. int a=Integer.parseInt(String.valueOf(c));
4. System.out.println(a);
5. }}

OUTPUT

1

**31.Write a program to print the pattern given below (Pyramid Star Pattern)**

**public** **class** PyramidStarOattern {

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

//i for rows and j for columns

//row denotes the number of rows you want to print

**int** i, j, row = 6;

//Outer loop work for rows

**for** (i=0; i<row; i++)

{

//inner loop work for space

**for** (j=row-i; j>1; j--)

{

//prints space between two stars

System.***out***.print(" ");

}

//inner loop for columns

**for** (j=0; j<=i; j++ )

{

//prints star

System.***out***.print("\* ");

}

//throws the cursor in a new line after printing each line

System.***out***.println();

}

}

}

OUTPUT:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

**32.Write a program to print Fibonacci Series up to count 10.**

**package** com.hello;

**public** **class** FebonacciSeries {

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

**int** n1=0,n2=1,n3,i,count=10;

System.***out***.print(n1+" "+n2);//printing 0 and 1

**for**(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed

{

n3=n1+n2;

System.***out***.print(" "+n3);

n1=n2;

n2=n3;

}

}

}

OUTPUT: 0 1 1 2 3 5 8 13 21 34

**33.How to reverse a String in Java?**

## 1) By StringBuilder / StringBuffer

*File: StringFormatter.java*

1. **public** **class** StringFormatter {
2. **public** **static** String reverseString(String str){
3. StringBuilder sb=**new** StringBuilder(str);
4. sb.reverse();
5. **return** sb.toString();
6. }
7. }
8. *File: TestStringFormatter.java*

**public** **class** TestStringFormatter {

1. **public** **static** **void** main(String[] args) {
2. System.out.println(StringFormatter.reverseString("my name is khan"));
3. System.out.println(StringFormatter.reverseString("I am sonoo jaiswal"));
4. }
5. }

Output: nahk si eman ym

lawsiaj oonos ma I

**34. .How To Find the Largest Value from the Given Array**

**package** com.hello;

**public** **class** FindMaxArray {

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

// **TODO** Auto-generated method stub

**int** [] arr = {1, 2100, 3, 455, 5, 34, 67};

**int** max = Integer.***MIN\_VALUE***;

**for**(**int** e: arr){

**if**(e>max){

max = e;

}

}

System.***out***.println("the value of the maximum element in this array is: "+ max);

}

}

OUTPUT:

the value of the maximum element in this array is: 2100

**35. How to display all the prime numbers between 1 and 100**

**package** com.hello;

**import** java.util.ArrayList;

**public** **class** PrimeNmbae {

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

ArrayList<Integer> a=**new** ArrayList<>();

**for**(**int** n=1; n<=100; n++){

**int** c = 0;

**for** (**int** i = 1; i <= n; i++)

**if** (n % i == 0)

c++;

**if** (c == 2)

a.add(n);

**else**

**continue**;

}

System.***out***.println(a);

}

}

**OUTPUT:** [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

**36. How to display all the prime numbers between 1 and n (n is the number, get the input from user)**

**package** com.hello;

**import** java.util.Scanner;

**public** **class** PrimeNumber1toN {

**private** **static** Scanner *scanner* = **new** Scanner( System.***in*** );

**public** **class** PrimeNumbet1toN {

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

String maxNumber = **null**;

System.***out***.println("Enter max number: "+ maxNumber);

String input = *scanner*.nextLine();

**int** maxNumber1 = Integer.*parseInt*( input );

System.***out***.println("List of the prime number between 1 - N" + maxNumber1);

**for** (**int** num = 2; num <= maxNumber1; num++)

{

**boolean** isPrime = **true**;

**for** (**int** i=2; i <= num/2; i++)

{

**if** ( num % i == 0)

{

isPrime = **false**;

**break**;

}

}

**if** ( isPrime == **true** )

System.***out***.println(num);

}

}

}

}

OUTPUT: Enter max number:

100

List of the prime number between 1 - 100

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

1. **37. How to find the given number is a prime number or not by getting input from the user**
2. **int** i,m=0,flag=0;
3. **int** n=3;//it is the number to be checked
4. m=n/2;
5. **if**(n==0||n==1){
6. System.out.println(n+" is not prime number");
7. }**else**{
8. **for**(i=2;i<=m;i++){
9. **if**(n%i==0){
10. System.out.println(n+" is not prime number");
11. flag=1;
12. **break**;
13. }
14. }
15. **if**(flag==0)  { System.out.println(n+" is prime number"); }
16. }//end of else
17. }
18. }

OUTPUT:5 is prime number

**38 .How to read a file line by line in Java?**

There are following ways to read a file line by line.

* BufferedReader Class
* Scanner class

## Using BufferedReader Class

Using the Java BufferedRedaer class is the most common and simple way to read a file line by line in Java. It belongs to **java.io** package. Java BufferedReader class provides readLine() method to read a file line by line. The signature of the method is:

**public** String readLine() **throws** IOException

## Using the Scanner class

Java **Scanner** class provides more utility methods compare to BufferedReader class. Java Scanner class provides the nextLine() method to facilitates line by line of file's content. The nextLine() methods returns the same String as readLine() method. The Scanner class can also read a file form InputStream.

**39. Difference between Array and ArrayList?**

|  |  |  |
| --- | --- | --- |
| Basis | Array | ArrayList |
| Definition | An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location. | The **ArrayList** is a class of Java **Collections** framework. It contains popular classes like **Vector, HashTable**, and **HashMap**. |
| Static/ Dynamic | Array is **static** in size. | ArrayList is **dynamic** in size. |
| Resizable | An array is a **fixed-length** data structure. | ArrayList is a **variable-length** data structure. It can be resized itself when needed. |
| Initialization | It is mandatory to provide the size of an array while initializing it directly or indirectly. | We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size. |
| Performance | It performs **fast** in comparison to ArrayList because of fixed size. | ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance. |
| Primitive/ Generic type | An array can store both **objects** and **primitives** type. | We cannot store **primitive** type in ArrayList. It automatically converts primitive type to object. |
| Iterating Values | We use **for** loop or **for each** loop to iterate over an array. | We use an **iterator** to iterate over ArrayList. |
| Type-Safety | We cannot use generics along with array because it is not a convertible type of array. | ArrayList allows us to store only **generic/ type, that's why it is type-safe.** |
| Length | Array provides a **length** variable which denotes the length of an array. | ArrayList provides the **size()** method to determine the size of ArrayList. |
| Adding Elements | We can add elements in an array by using the **assignment**operator. | Java provides the **add()** method to add elements in the ArrayList. |
| Single/ Multi-Dimensional | Array can be **multi-dimensional**. | ArrayList is always **single-dimensional**. |

**40. .Difference between ArrayList and HashSet in Java?**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Key | ArrayList | HashSet |
| 1 | Implementation | ArrayList is the implementation of the list interface. | HashSet on the other hand is the implementation of a set interface. |
| 2 | Internal implementation | ArrayList internally implements array for its implementation. | HashSet internally uses Hashmap for its implementation. |
| 3 | Order of elements | ArrayList maintains the insertion order i.e. order of the object in which they are inserted. | HashSet is an unordered collection and doesn't maintain any order. |
| 4 | Duplicates | ArrayList allows duplicate values in its collection. | On other hand duplicate elements are not allowed in Hashset. |
| 5 | Index performance | ArrayList uses index for its performance i.e. its index based one can retrieve object by calling get(index) or remove objects by calling remove(index) | HashSet is completely based on object also it doesn't provide get() method. |
| 6 | Null Allowed | Any number of null value can be inserted in ArrayList without any restriction. | On other hand Hashset allows only one null value in its collection, after which no null value is allowed to be added. |

**41. What are the different access modifiers available in Java?**

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.

**42. Difference between static binding and dynamic binding?**

|  |  |
| --- | --- |
| Static Binding | Dynamic Binding |
| It takes place at compile time for which is referred to as early binding | It takes place at runtime so do it is referred to as late binding. |
| It uses overloading more precisely operator overloading method | It uses overriding methods. |
| It takes place using normal functions | It takes place using virtual functions |
| Real objects never use static binding | Real objects use dynamic binding. |

**43 Difference between Abstract Class and Interface?**

|  |  |
| --- | --- |
| Abstract class | Interface |
| 1) 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. |
| 2) Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| 3) Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| 4) Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class. |
| 5) The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| 6) An abstract class can extend another Java class and implement multiple Java interfaces. | An interface can extend another Java interface only. |
| 7) An abstract class can be extended using keyword "extends". | An interface can be implemented using keyword "implements". |
| 8) A Java abstract class can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)Example: public abstract class Shape{ public abstract void draw(); } | Example: public interface Drawable{ void draw(); } |

**44.What is Multiple Inheritance?**

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

**45. What are the differences between throw and throws in Java?**

|  |  |  |  |
| --- | --- | --- | --- |
| 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. |

**46. When you can use abstract class and interface in java**

## When to use an abstract class

* An abstract class is a good choice if we are using the inheritance concept since it provides a common base class implementation to derived classes.
* An abstract class is also good if we want to declare non-public members. In an interface, all methods must be public.
* If we want to add new methods in the future, then an abstract class is a better choice. Because if we add new methods to an interface, then all of the classes that already implemented that interface will have to be changed to implement the new methods.
* abstract class Car {
* public void accelerate() {
* System.out.println("Do something to accelerate");
* }
* public void applyBrakes() {
* System.out.println("Do something to apply brakes");
* }
* public abstract void changeGears();
* }
* Now, any Car that wants to be instantiated must implement the changeGears () method.
* class Alto extends Car {
* public void changeGears() {
* System.out.println("Implement changeGears() method for Alto Car");
* }
* }
* class Santro extends Car {
* public void changeGears() {
* System.out.println("Implement changeGears() method for Santro Car");
* } }

## When to use an interface

* If the functionality we are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing a common functionality to unrelated classes.
* Interfaces are a good choice when we think that the API will not change for a while.
* Interfaces are also good when we want to have something similar to multiple inheritances since we can implement multiple interfaces.

## Example

public interface Actor {

   void perform();

}

public interface Producer {

   void invest();

}

Now a days most of the actors are rich enough to produce their own movie. If we are using interfaces rather than abstract classes, we can implement both Actor and Producer. Also, we can define a new ActorProducer interface that extends both.

public interface ActorProducer extends Actor, Producer{

   // some statements

}

Java Interview Programs

**Java Collection Programs**

**1.Java Program to travel collection?**

In Java, **Travelling Salesman Problem** is a problem in which we need to find the shortest route that covers each city exactly once and returns to the starting point. Hamiltonian Cycle is another problem in Java that is mostly similar to Travelling Salesman Problem.

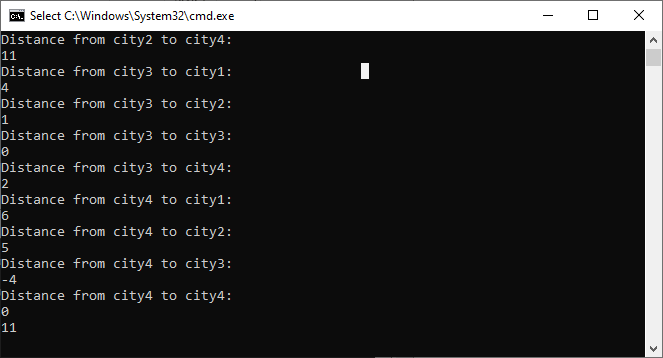
The main difference between TSP and the Hamiltonian cycle is that in Hamiltonian Cycle, we are not sure whether a tour that visits each city exactly once exists or not, and we have to determine it. In the Travelling Salesman Problem, a Hamiltonian cycle is always present because the graph is complete, and the problem is to find a Hamiltonian cycle with minimum weight.

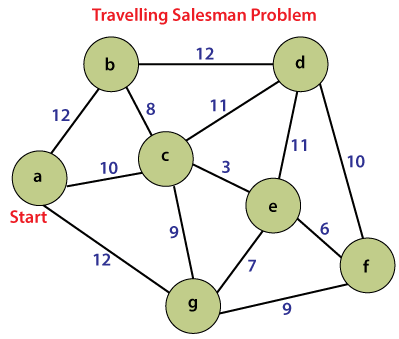
Let's take an example of the Travelling Salesman Problem to understand the problem.

1. **import** Java.util.\*;
2. **import** java.io.\*;
3. **import** java.util.Scanner;
5. // create TSPExample class to implement TSP code in Java
6. **class** TSPExample
7. {
8. // create findHamiltonianCycle() method to get minimum weighted cycle
9. **static** **int** findHamiltonianCycle(**int**[][] distance, **boolean**[] visitCity, **int** currPos, **int** cities, **int** count, **int** cost, **int** hamiltonianCycle)
10. {
12. **if** (count == cities && distance[currPos][0] > 0)
13. {
14. hamiltonianCycle = Math.min(hamiltonianCycle, cost + distance[currPos][0]);
15. **return** hamiltonianCycle;
16. }
18. // BACKTRACKING STEP
19. **for** (**int** i = 0; i < cities; i++)
20. {
21. **if** (visitCity[i] == **false** && distance[currPos][i] > 0)
22. {
24. // Mark as visited
25. visitCity[i] = **true**;
26. hamiltonianCycle = findHamiltonianCycle(distance, visitCity, i, cities, count + 1, cost + distance[currPos][i], hamiltonianCycle);
28. // Mark ith node as unvisited
29. visitCity[i] = **false**;
30. }
31. }
32. **return** hamiltonianCycle;
33. }
35. // main() method start
36. **public** **static** **void** main(String[] args)
37. {
38. **int** cities;
40. //create scanner class object to get input from user
41. Scanner sc = **new** Scanner(System.in);
43. // get total number of cities from the user
44. System.out.println("Enter total number of cities ");
45. cities = sc.nextInt();

48. //get distance of cities from the user
49. **int** distance[][] = **new** **int**[cities][cities];
50. **for**( **int** i = 0; i < cities; i++){
51. **for**( **int** j = 0; j < cities; j++){
52. System.out.println("Distance from city"+ (i+1) +" to city"+ (j+1) +": ");
53. distance[i][j] = sc.nextInt();
54. }
55. }
57. // create an array of type boolean to check if a node has been visited or not
58. **boolean**[] visitCity = **new** **boolean**[cities];
60. // by default, we make the first city visited
61. visitCity[0] = **true**;

64. **int** hamiltonianCycle = Integer.MAX\_VALUE;
66. // call findHamiltonianCycle() method that returns the minimum weight Hamiltonian Cycle
67. hamiltonianCycle = findHamiltonianCycle(distance, visitCity, 0, cities, 1, 0, hamiltonianCycle);
69. // print the minimum weighted Hamiltonian Cycle
70. System.out.println(hamiltonianCycle);
71. }
72. }

**Output**  




In the above graph, **a, b, c, d, e, f**, and **g** are the cities and we have to find the Hamiltonian cycle having minimum weight

**2. Java Program to for Array List and HashMap?**

### Iterating ArrayList using Iterator

### import java.util.\*;

1. **public** **class** ArrayListExample2{
2. **public** **static** **void** main(String args[]){
3. ArrayList<String> list=**new** ArrayList<String>();//Creating arraylist
4. list.add("Mango");//Adding object in arraylist
5. list.add("Apple");
6. list.add("Banana");
7. list.add("Grapes");
8. //Traversing list through Iterator
9. Iterator itr=list.iterator();//getting the Iterator
10. **while**(itr.hasNext()){//check if iterator has the elements
11. System.out.println(itr.next());//printing the element and move to next
12. }
13. }
14. }

**OUTPUT:**

Mango

Apple

Banana

Grapes

### Java HashMap

1. **import** java.util.\*;
2. **public** **class** HashMapExample1{
3. **public** **static** **void** main(String args[]){
4. HashMap<Integer,String> map=**new** HashMap<Integer,String>();//Creating HashMap
5. map.put(1,"Mango");  //Put elements in Map
6. map.put(2,"Apple");
7. map.put(3,"Banana");
8. map.put(4,"Grapes");
10. System.out.println("Iterating Hashmap...");
11. **for**(Map.Entry m : map.entrySet()){
12. System.out.println(m.getKey()+" "+m.getValue());
13. }
14. }
15. }

OUTPUT: Iterating Hashmap...

1 Mango

2 Apple

3 Banana

* 1. Grapes

**3.Java Program to travel hash map collection?**

**import** java.util.\*;

**public** **class** HashMapExample1{

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

   HashMap<Integer,String> map=**new** HashMap<Integer,String>();//Creating HashMap

    map.put(1,"Mango");  //Put elements in Map     map.put(2,"Apple");

   map.put(3,"Banana");

   map.put(4,"Grapes");

    System.out.println("Iterating Hashmap...");

**for**(Map.Entry m : map.entrySet()){

     System.out.println(m.getKey()+" "+m.getValue());

    }

}

}

OUTPUT: Iterating Hashmap...

1 Mango

2 Apple

3 Banana

1. **Java Program to elaborate comparator and comparable in collection?**

|  |  |
| --- | --- |
| Comparable | Comparator |
| 1) Comparable provides a single sorting sequence. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable affects the original class, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides compareTo() method to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in java.lang package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by Collections.sort(List) method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

## Java Comparable Example

1. //Java Program to demonstrate the use of Java Comparable.
2. //Creating a class which implements Comparable Interface
3. **import** java.util.\*;
4. **import** java.io.\*;
5. **class** Student **implements** Comparable<Student>{
6. **int** rollno;
7. String name;
8. **int** age;
9. Student(**int** rollno,String name,**int** age){
10. **this**.rollno=rollno;
11. **this**.name=name;
12. **this**.age=age;
13. }
14. **public** **int** compareTo(Student st){
15. **if**(age==st.age)
16. **return** 0;
17. **else** **if**(age>st.age)
18. **return** 1;
19. **else**
20. **return** -1;
21. }
22. }
23. //Creating a test class to sort the elements
24. **public** **class** TestSort3{
25. **public** **static** **void** main(String args[]){
26. ArrayList<Student> al=**new** ArrayList<Student>();
27. al.add(**new** Student(101,"Vijay",23));
28. al.add(**new** Student(106,"Ajay",27));
29. al.add(**new** Student(105,"Jai",21));
31. Collections.sort(al);
32. **for**(Student st:al){
33. System.out.println(st.rollno+" "+st.name+" "+st.age);
34. }
35. }
36. }

## Java Comparator Example

Let's see an example of the Java Comparator interface where we are sorting the elements of a list using different comparators.

**Student.java**

1. **class** Student{
2. **int** rollno;
3. String name;
4. **int** age;
5. Student(**int** rollno,String name,**int** age){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.age=age;
9. }
10. }

**AgeComparator.java**

1. **import** java.util.\*;
2. **class** AgeComparator **implements** Comparator<Student>{
3. **public** **int** compare(Student s1,Student s2){
4. **if**(s1.age==s2.age)
5. **return** 0;
6. **else** **if**(s1.age>s2.age)
7. **return** 1;
8. **else**
9. **return** -1;
10. }
11. }

**NameComparator.java**

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

1. **import** java.util.\*;
2. **class** NameComparator **implements** Comparator<Student>{
3. **public** **int** compare(Student s1,Student s2){
4. **return** s1.name.compareTo(s2.name);
5. }
6. }

**TestComparator.java**

In this class, we are printing the values of the object by sorting on the basis of name and age.

1. //Java Program to demonstrate the use of Java Comparator
2. **import** java.util.\*;
3. **import** java.io.\*;
4. **class** TestComparator{
5. **public** **static** **void** main(String args[]){
6. //Creating a list of students
7. ArrayList<Student> al=**new** ArrayList<Student>();
8. al.add(**new** Student(101,"Vijay",23));
9. al.add(**new** Student(106,"Ajay",27));
10. al.add(**new** Student(105,"Jai",21));
12. System.out.println("Sorting by Name");
13. //Using NameComparator to sort the elements
14. Collections.sort(al,**new** NameComparator());
15. //Traversing the elements of list
16. **for**(Student st: al){
17. System.out.println(st.rollno+" "+st.name+" "+st.age);
18. }
20. System.out.println("sorting by Age");
21. //Using AgeComparator to sort the elements
22. Collections.sort(al,**new** AgeComparator());
23. //Travering the list again
24. **for**(Student st: al){
25. System.out.println(st.rollno+" "+st.name+" "+st.age);
26. }
28. }
29. }

Output:

Sorting by Name

106 Ajay 27

105 Jai 21

101 Vijay 23

Sorting by Age

105 Jai 21

101 Vijay 23

106 Ajay 27

**Java String Programs**

**1.Java Program to count the total number of characters in a string**

1. **public** **class** CountCharacter
2. {
3. **public** **static** **void** main(String[] args) {
4. String string = "The best of both worlds";
5. **int** count = 0;
7. //Counts each character except space
8. **for**(**int** i = 0; i < string.length(); i++) {
9. **if**(string.charAt(i) != ' ')
10. count++;
11. }
13. //Displays the total number of characters present in the given string
14. System.out.println("Total number of characters in a string: " + count);
15. }
16. }

**2.Java Program to count the total number of vowels and consonants in a string.**

1. **public** **class** CountVowelConsonant {
2. **public** **static** **void** main(String[] args) {
4. //Counter variable to store the count of vowels and consonant
5. **int** vCount = 0, cCount = 0;
7. //Declare a string
8. String str = "This is a really simple sentence";
10. //Converting entire string to lower case to reduce the comparisons
11. str = str.toLowerCase();
13. **for**(**int** i = 0; i < str.length(); i++) {
14. //Checks whether a character is a vowel
15. **if**(str.charAt(i) == 'a' || str.charAt(i) == 'e' || str.charAt(i) == 'i' || str.charAt(i) == 'o' || str.charAt(i) == 'u') {
16. //Increments the vowel counter
17. vCount++;
18. }
19. //Checks whether a character is a consonant
20. **else** **if**(str.charAt(i) >= 'a' && str.charAt(i)<='z') {
21. //Increments the consonant counter
22. cCount++;
23. }
24. }
25. System.out.println("Number of vowels: " + vCount);
26. System.out.println("Number of consonants: " + cCount);
27. }
28. }

**Output:**

Number of vowels: 10

Number of consonants: 17

**3.Java Program to remove all the white spaces from a string**

1. **ublic** **class** removeWhiteSpace {
2. **public** **static** **void** main(String[] args) {
4. String str1="Remove white spaces";
6. //Removes the white spaces using regex
7. str1 = str1.replaceAll("\\s+", "");
9. System.out.println("String after removing all the white spaces : " + str1);
10. }
11. }

**Output:**

29.7M

546

Prime Ministers of India | List of Prime Minister of India (1947-2020)

String after removing all the white spaces: Removewhitespaces

**4.Java Program to replace lower-case characters with upper-case and vice-versa .**

1. **public** **class** changeCase {
2. **public** **static** **void** main(String[] args) {
4. String str1="Great Power";
5. StringBuffer newStr=**new** StringBuffer(str1);
7. **for**(**int** i = 0; i < str1.length(); i++) {
9. //Checks for lower case character
10. **if**(Character.isLowerCase(str1.charAt(i))) {
11. //Convert it into upper case using toUpperCase() function
12. newStr.setCharAt(i, Character.toUpperCase(str1.charAt(i)));
13. }
14. //Checks for upper case character
15. **else** **if**(Character.isUpperCase(str1.charAt(i))) {
16. //Convert it into upper case using toLowerCase() function
17. newStr.setCharAt(i, Character.toLowerCase(str1.charAt(i)));
18. }
19. }
20. System.out.println("String after case conversion : " + newStr);
21. }
22. }

**Output:**

String after case conversion: gREAT pOWER

**5. Java Program to replace the spaces of a string with a specific character**

1. **public** **class** ReplaceSpace
2. {
3. **public** **static** **void** main(String[] args) {
4. String string = "Once in a blue moon";
5. **char** ch = '-';
7. //Replace space with specific character ch
8. string = string.replace(' ', ch);
10. System.out.println("String after replacing spaces with given character: ");
11. System.out.println(string);
12. }
13. }

**Output:**

String after replacing spaces with given character:

Once-in-a-blue-moon

**6.Java Program to determine whether a given string is palindrome**

1. **public** **class** PalindromeString
2. {
3. **public** **static** **void** main(String[] args) {
4. String string = "Kayak";
5. **boolean** flag = **true**;
7. //Converts the given string into lowercase
8. string = string.toLowerCase();
10. //Iterate the string forward and backward, compare one character at a time
11. //till middle of the string is reached
12. **for**(**int** i = 0; i < string.length()/2; i++){
13. **if**(string.charAt(i) != string.charAt(string.length()-i-1)){
14. flag = **false**;
15. **break**;
16. }
17. }
18. **if**(flag)
19. System.out.println("Given string is palindrome");
20. **else**
21. System.out.println("Given string is not a palindrome");
22. }
23. }

**Output:**

Given string is palindrome

**7.Java Program to find maximum and minimum occurring character in a string**

### JAVA

1. **public** **class** Characters
2. {
3. **public** **static** **void** main(String[] args) {
4. String str = "grass is greener on the other side";
5. **int**[] freq = **new** **int**[str.length()];
6. **char** minChar = str.charAt(0), maxChar = str.charAt(0);
7. **int** i, j, min, max;
9. //Converts given string into character array
10. **char** string[] = str.toCharArray();
12. //Count each word in given string and store in array freq
13. **for**(i = 0; i < string.length; i++) {
14. freq[i] = 1;
15. **for**(j = i+1; j < string.length; j++) {
16. **if**(string[i] == string[j] && string[i] != ' ' && string[i] != '0') {
17. freq[i]++;
19. //Set string[j] to 0 to avoid printing visited character
20. string[j] = '0';
21. }
22. }
23. }
25. //Determine minimum and maximum occurring characters
26. min = max = freq[0];
27. **for**(i = 0; i <freq.length; i++) {
29. //If min is greater than frequency of a character
30. //then, store frequency in min and corresponding character in minChar
31. **if**(min > freq[i] && freq[i] != '0') {
32. min = freq[i];
33. minChar = string[i];
34. }
35. //If max is less than frequency of a character
36. //then, store frequency in max and corresponding character in maxChar
37. **if**(max < freq[i]) {
38. max = freq[i];
39. maxChar = string[i];
40. }
41. }
43. System.out.println("Minimum occurring character: " + minChar);
44. System.out.println("Maximum occurring character: " + maxChar);
45. }
46. }

**Output:**

Minimum occurring character: a

Maximum occurring character: e

**8.Java Program to Reverse the string.**

1. **public** **class** Reverse
2. {
3. **public** **static** **void** main(String[] args) {
4. String string = "Dream big";
5. //Stores the reverse of given string
6. String reversedStr = "";
8. //Iterate through the string from last and add each character to variable reversedStr
9. **for**(**int** i = string.length()-1; i >= 0; i--){
10. reversedStr = reversedStr + string.charAt(i);
11. }
13. System.out.println("Original string: " + string);
14. //Displays the reverse of given string
15. System.out.println("Reverse of given string: " + reversedStr);

    }    }

**OUTPUT:** Original string: Dream big

Reverse of given string: gib maerD

1. **Java program to find duplicate characters in a string.**
2. **public** **class** DuplicateCharacters {
3. **public** **static** **void** main(String[] args) {
4. String string1 = "Great responsibility";
5. **int** count;
7. //Converts given string into character array
8. **char** string[] = string1.toCharArray();
10. System.out.println("Duplicate characters in a given string: ");
11. //Counts each character present in the string
12. **for**(**int** i = 0; i <string.length; i++) {
13. count = 1;
14. **for**(**int** j = i+1; j <string.length; j++) {
15. **if**(string[i] == string[j] && string[i] != ' ') {
16. count++;
17. //Set string[j] to 0 to avoid printing visited character
18. string[j] = '0';
19. }
20. }
21. //A character is considered as duplicate if count is greater than 1
22. **if**(count > 1 && string[i] != '0')
23. System.out.println(string[i]);
24. }
25. }
26. }

**Output:**

Duplicate characters in a given string:

r

e

t

s

i

**10.Java program to find the duplicate words in a string .**

1. **public** **class** DuplicateWord {
2. **public** **static** **void** main(String[] args) {
3. String string = "Big black bug bit a big black dog on his big black nose";
4. **int** count;
6. //Converts the string into lowercase
7. string = string.toLowerCase();
9. //Split the string into words using built-in function
10. String words[] = string.split(" ");
12. System.out.println("Duplicate words in a given string : ");
13. **for**(**int** i = 0; i < words.length; i++) {
14. count = 1;
15. **for**(**int** j = i+1; j < words.length; j++) {
16. **if**(words[i].equals(words[j])) {
17. count++;
18. //Set words[j] to 0 to avoid printing visited word
19. words[j] = "0";
20. }
21. }
23. //Displays the duplicate word if count is greater than 1
24. **if**(count > 1 && words[i] != "0")
25. System.out.println(words[i]);
26. }
27. }
28. }

**Output:**

Duplicate words in a given string :

big

black

**11.Java Program to find the frequency of characters?**

1. **public** **class** Frequency
2. {
3. **public** **static** **void** main(String[] args) {
4. String str = "picture perfect";
5. **int**[] freq = **new** **int**[str.length()];
6. **int** i, j;
8. //Converts given string into character array
9. **char** string[] = str.toCharArray();
11. **for**(i = 0; i <str.length(); i++) {
12. freq[i] = 1;
13. **for**(j = i+1; j <str.length(); j++) {
14. **if**(string[i] == string[j]) {
15. freq[i]++;
17. //Set string[j] to 0 to avoid printing visited character
18. string[j] = '0';
19. }
20. }
21. }
23. //Displays the each character and their corresponding frequency
24. System.out.println("Characters and their corresponding frequencies");
25. **for**(i = 0; i <freq.length; i++) {
26. **if**(string[i] != ' ' && string[i] != '0')
27. System.out.println(string[i] + "-" + freq[i]);
28. }
29. }
30. }

**Output:**

Characters and their corresponding frequencies

p-2

i-1

c-2

t-2

u-1

r-2

e-3

f-1

1. Java Program to find the maximum and minimum occurring word count in a string?
2. **public** **class** Characters
3. {
4. **public** **static** **void** main(String[] args) {
5. String str = "grass is greener on the other side";
6. **int**[] freq = **new** **int**[str.length()];
7. **char** minChar = str.charAt(0), maxChar = str.charAt(0);
8. **int** i, j, min, max;
10. //Converts given string into character array
11. **char** string[] = str.toCharArray();
13. //Count each word in given string and store in array freq
14. **for**(i = 0; i < string.length; i++) {
15. freq[i] = 1;
16. **for**(j = i+1; j < string.length; j++) {
17. **if**(string[i] == string[j] && string[i] != ' ' && string[i] != '0') {
18. freq[i]++;
20. //Set string[j] to 0 to avoid printing visited character
21. string[j] = '0';
22. }
23. }
24. }
26. //Determine minimum and maximum occurring characters
27. min = max = freq[0];
28. **for**(i = 0; i <freq.length; i++) {
30. //If min is greater than frequency of a character
31. //then, store frequency in min and corresponding character in minChar
32. **if**(min > freq[i] && freq[i] != '0') {
33. min = freq[i];
34. minChar = string[i];
35. }
36. //If max is less than frequency of a character
37. //then, store frequency in max and corresponding character in maxChar
38. **if**(max < freq[i]) {
39. max = freq[i];
40. maxChar = string[i];
41. }
42. }
44. System.out.println("Minimum occurring character: " + minChar);
45. System.out.println("Maximum occurring character: " + maxChar);
46. }
47. }

**Output:**

Minimum occurring character: a

Maximum occurring character: e

**12.Java Program to find the total number of the words in the given string?**

1. **public** **class** WordCount {
2. **static** **int** wordcount(String string)
3. {
4. **int** count=0;
6. **char** ch[]= **new** **char**[string.length()];
7. **for**(**int** i=0;i<string.length();i++)
8. {
9. ch[i]= string.charAt(i);
10. **if**( ((i>0)&&(ch[i]!=' ')&&(ch[i-1]==' ')) || ((ch[0]!=' ')&&(i==0)) )
11. count++;
12. }
13. **return** count;
14. }
15. **public** **static** **void** main(String[] args) {
16. String string ="    India Is My Country";
17. System.out.println(wordcount(string) + " words.");
18. }
19. }

Output:

4 words.

**13. Java Program to separate the Individual Characters from a String?**

1. **public** **class** IndividualCharacters
2. {
3. **public** **static** **void** main(String[] args) {
4. String string = "characters";
6. //Displays individual characters from given string
7. System.out.println("Individual characters from given string:");
9. //Iterate through the string and display individual character
10. **for**(**int** i = 0; i < string.length(); i++){
11. System.out.print(string.charAt(i) + "  ");
12. }
13. }
14. }

**Output:**

Individual characters from given string:

c h a r a c t e r s

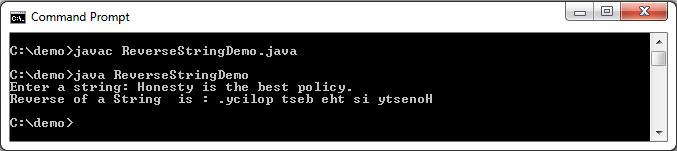
**14.Java Program to swap two string variables without using third or temp variable ?**

1. **public** **class** SwapWithoutTemp {
2. **public** **static** **void** main(String args[]) {
3. String a = "Love";
4. String b = "You";
5. System.out.println("Before swap: " + a + " " + b);
6. a = a + b;
7. b = a.substring(0, a.length() - b.length());
8. a = a.substring(b.length());
9. System.out.println("After : " + a + " " + b);
10. }
11. }

**15.Java Program to Reverse String in Java Word by Word (sentence)?**

1. **import** java.util.\*;
2. **public** **class** ReverseStringDemo
3. {
4. **public** **static** **void** main(String[] arg)
5. {
6. ReverseStringDemo rs=**new** ReverseStringDemo();
7. Scanner sc=**new** Scanner(System.in);
8. System.out.print("Enter a string: ");
9. String  str=sc.nextLine();
10. System.out.println("Reverse of a String  is : "+rs.reversestr(str));          //called method
11. }
12. //reverse string method
13. **static** String reversestr(String s)
14. {
15. String r="";
16. **for**(**int** i=s.length();i>0;--i)        //execute until condition i>0 becomes false
17. {
18. r=r+(s.charAt(i-1));
19. }
20. **return** r;
21. }
22. }

**Output:**



**16. Java Program to Reserve String without reverse () function**

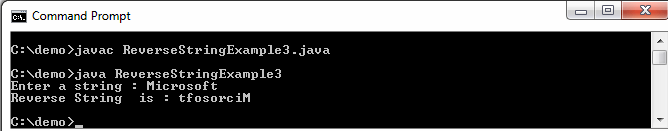
### Using Static method

**Example to reverse string in Java by using static method**

In the following example, we have created an object of the class and called the static method with the object as rev.reverse(s) by passing the given string.

1. **import** java.util.Scanner;
2. **public** **class** ReverseStringExample3
3. {
4. **public** **static** **void** main(String[] arg)
5. {
6. ReverseStringExample3 rev=**new** ReverseStringExample3();
7. Scanner sc=**new** Scanner(System.in);
8. System.out.print("Enter a string : ");
9. String  s=sc.nextLine();
10. System.out.println("Reverse String  is : "+rev.reverse(s)); //calling method
11. }
12. //calling method
13. **static** String reverse(String str)
14. {
15. String rev="";
16. **for**(**int** i=str.length();i>0;--i)
17. {
18. rev=rev+(str.charAt(i-1));
19. }
20. **return** rev;
21. }
22. }

**Output:**



**17. Java Program to Display first letter of words in sentence as capital**

1. **public** **class** StringFormatter {
2. **public** **static** String capitalizeWord(String str){
3. String words[]=str.split("\\s");
4. String capitalizeWord="";
5. **for**(String w:words){
6. String first=w.substring(0,1);
7. String afterfirst=w.substring(1);
8. capitalizeWord+=first.toUpperCase()+afterfirst+" ";
9. }
10. **return** capitalizeWord.trim();
11. }
12. }

*File: TestStringFormatter.java*

17.2M

263

Triggers in SQL (Hindi)

1. **public** **class** TestStringFormatter {
2. **public** **static** **void** main(String[] args) {
3. System.out.println(StringFormatter.capitalizeWord("my name is khan"));
4. System.out.println(StringFormatter.capitalizeWord("I am sonoo jaiswal"));
5. }
6. }

Output:

My Name Is Khan

I Am Sonoo Jaiswal

**18.Java Program to find length of string without built in functions?**

1. public class StringLength {
2. public static void main(String args[]) throws Exception {
3. String str = "sampleString";
4. int i = 0;
5. for(char c: str.toCharArray()) {
6. i++;
7. }
8. System.out.println("Length of the given string ::"+i);
9. }
10. }

## Output

1. Length of the given string ::12

**19. Java Program to concatenate two strings?**

### 2) String Concatenation by concat() method

The String concat() method concatenates the specified string to the end of current string. Syntax:

1. **public** String concat(String another)

Let's see the example of String concat() method.

**TestStringConcatenation3.java**

1. **class** TestStringConcatenation3{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin ";
4. String s2="Tendulkar";
5. String s3=s1.concat(s2);
6. System.out.println(s3);//Sachin Tendulkar
7. }
8. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation3)

**Output:**

Sachin Tendulkar

**20.Java Program to display number of spaces in sentence?**

1. //program to count te space of the given string
2. import java.util.Scanner;
3. public class Count\_space{
4. public static void main(String args[]){
5. //variable declaration
6. String str;
7. int space=0;
8. //vriable declaration and initialization
9. Scanner scan=new Scanner(System.in);
10. //create a scanner object for input
12. System.out.println("Enter the String ");
13. str=scan.nextLine();
14. for(int i=0; i<=str.length(); i++){
15. char ch=str.charAt(i);
16. if(ch==' '){
17. space++;
18. }
19. }
20. System.out.println("Total white space : "+space);
21. }
22. }

OUTPUT:

1. Enter the string
2. code4coding.com website provides Java tutorials for beginners
3. Total white space: 6

**21.Java Program to display duplicate word and character from string?**

1. **public** **class** DuplicateWord {
2. **public** **static** **void** main(String[] args) {
3. String string = "Big black bug bit a big black dog on his big black nose";
4. **int** count;
6. //Converts the string into lowercase
7. string = string.toLowerCase();
9. //Split the string into words using built-in function
10. String words[] = string.split(" ");
12. System.out.println("Duplicate words in a given string : ");
13. **for**(**int** i = 0; i < words.length; i++) {
14. count = 1;
15. **for**(**int** j = i+1; j < words.length; j++) {
16. **if**(words[i].equals(words[j])) {
17. count++;
18. //Set words[j] to 0 to avoid printing visited word
19. words[j] = "0";
20. }
21. }
23. //Displays the duplicate word if count is greater than 1
24. **if**(count > 1 && words[i] != "0")
25. System.out.println(words[i]);
26. }
27. }
28. }

**Output:**

7.2M

136

Duplicate words in a given string :

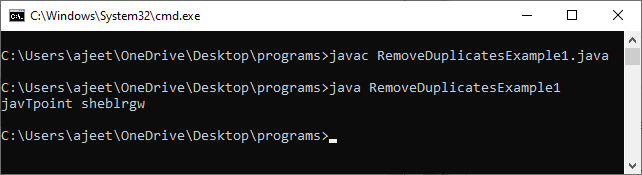
big

black

**22. Java Program to remove all duplicate characters from string?**

1. //Import packages
2. **import** java.util.\*;
4. //Creating RemoveDuplicatesExample1 class
5. **class** RemoveDuplicatesExample1
6. {
7. //Creating removeDuplicates() method to remove duplicates from array
8. **static** **void** removeDuplicate(**char** str[], **int** length)
9. {
10. //Creating index variable to use it as index in the modified string
11. **int** index = 0;
13. // Traversing character array
14. **for** (**int** i = 0; i < length; i++)
15. {
17. // Check whether str[i] is present before or not
18. **int** j;
19. **for** (j = 0; j < i; j++)
20. {
21. **if** (str[i] == str[j])
22. {
23. **break**;
24. }
25. }
27. // If the character is not present before, add it to resulting string
28. **if** (j == i)
29. {
30. str[index++] = str[i];
31. }
32. }
33. System.out.println(String.valueOf(Arrays.copyOf(str, index)));
34. }
36. // main() method starts
37. **public** **static** **void** main(String[] args)
38. {
39. String info = "javaTpoint is the best learning website";
40. //Converting string to character array
41. **char** str[] = info.toCharArray();
42. //Calculating length of the character array
43. **int** len = str.length;
44. //Calling removeDuplicates() method to remove duplicate characters
45. removeDuplicate(str, len);
46. }
47. }

**Output**



**23.Java Program to check string is Palindrome or not?**

|  |
| --- |
| 1. import java.util.Scanner; 2. class Palindrome 3. { 4. public static void main(String arg[]) 5. { 6. String str,strrev=""; 7. Scanner sc=new Scanner(System.in); 8. System.out.print("Enter a string :"); 9. str=sc.next(); 10. for(int i=str.length()-1;i>=0;i--) 11. { 12. strrev=strrev+str.charAt(i); 13. } 14. System.out.println("Reverse of a string is : "+strrev); 15. if(strrev.equalsIgnoreCase(str)) 16. System.out.println("Entered string is palindrome"); 17. else 18. System.out.println("Entered string is not a palindrome number "); 19. } 20. } |

**Output:**

|  |  |
| --- | --- |
| 1  2  3 | Enter a string : Dad  Reverse of a string is : daD  Entered string is palindrome |

Output

|  |  |
| --- | --- |
| 1  2  3 | Enter a string : Palindrome  Reverse of a string is : emordnilaP  Entered string is not a palindrome number |

**Java Array Programs**

1. Java Program to copy all elements of one array into another array

1. **public** **class** CopyArray {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr1 = **new** **int** [] {1, 2, 3, 4, 5};
5. //Create another array arr2 with size of arr1
6. **int** arr2[] = **new** **int**[arr1.length];
7. //Copying all elements of one array into another
8. **for** (**int** i = 0; i < arr1.length; i++) {
9. arr2[i] = arr1[i];
10. }
11. //Displaying elements of array arr1
12. System.out.println("Elements of original array: ");
13. **for** (**int** i = 0; i < arr1.length; i++) {
14. System.out.print(arr1[i] + " ");
15. }
17. System.out.println();
19. //Displaying elements of array arr2
20. System.out.println("Elements of new array: ");
21. **for** (**int** i = 0; i < arr2.length; i++) {
22. System.out.print(arr2[i] + " ");
23. }
24. }
25. }

**Output:**

Elements of original array

1 2 3 4 5

Elements of new array:

* 1. 2 3 4 5

**2.Java Program to find the frequency of each element in the array**

1. **public** **class** Frequency {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr = **new** **int** [] {1, 2, 8, 3, 2, 2, 2, 5, 1};
5. //Array fr will store frequencies of element
6. **int** [] fr = **new** **int** [arr.length];
7. **int** visited = -1;
8. **for**(**int** i = 0; i < arr.length; i++){
9. **int** count = 1;
10. **for**(**int** j = i+1; j < arr.length; j++){
11. **if**(arr[i] == arr[j]){
12. count++;
13. //To avoid counting same element again
14. fr[j] = visited;
15. }
16. }
17. **if**(fr[i] != visited)
18. fr[i] = count;
19. }
21. //Displays the frequency of each element present in array
22. System.out.println("---------------------------------------");
23. System.out.println(" Element | Frequency");
24. System.out.println("---------------------------------------");
25. **for**(**int** i = 0; i < fr.length; i++){
26. **if**(fr[i] != visited)
27. System.out.println("    " + arr[i] + "    |    " + fr[i]);
28. }
29. System.out.println("----------------------------------------");
30. }}

**Output:**

----------------------------------------

Element | Frequency

----------------------------------------

1 | 2

2 | 4

8 | 1

3 | 1

5 | 1

----------------------------------------

**3.Java Program to print the duplicate elements of an array.**

1. **public** **class** DuplicateElement {
2. **public** **static** **void** main(String[] args) {
4. //Initialize array
5. **int** [] arr = **new** **int** [] {1, 2, 3, 4, 2, 7, 8, 8, 3};
7. System.out.println("Duplicate elements in given array: ");
8. //Searches for duplicate element
9. **for**(**int** i = 0; i < arr.length; i++) {
10. **for**(**int** j = i + 1; j < arr.length; j++) {
11. **if**(arr[i] == arr[j])
12. System.out.println(arr[j]);
13. }
14. }
15. }
16. }

**Output:**

Duplicate elements in given array:

2

3

8

**4.Java Program to print the elements of an array**

1. **public** **class** PrintArray {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr = **new** **int** [] {1, 2, 3, 4, 5};
5. System.out.println("Elements of given array: ");
6. //Loop through the array by incrementing value of i
7. **for** (**int** i = 0; i < arr.length; i++) {
8. System.out.print(arr[i] + " ");
9. }
10. }
11. }

**Output:**

Elements of given array:

1 2 3 4 5

**5. Java Program to print the elements of an array in reverse order**

1. **public** **class** ReverseArray {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr = **new** **int** [] {1, 2, 3, 4, 5};
5. System.out.println("Original array: ");
6. **for** (**int** i = 0; i < arr.length; i++) {
7. System.out.print(arr[i] + " ");
8. }
9. System.out.println();
10. System.out.println("Array in reverse order: ");
11. //Loop through the array in reverse order
12. **for** (**int** i = arr.length-1; i >= 0; i--) {
13. System.out.print(arr[i] + " ");
14. }
15. }
16. }

**Output:**

Original array:

1 2 3 4 5

Array in reverse order:

5 4 3 2 1

**6.Java Program to print the elements of an array present on even position**

1. **public** **class** EvenPosition {
2. **public** **static** **void** main(String[] args) {
4. //Initialize array
5. **int** [] arr = **new** **int** [] {1, 2, 3, 4, 5};
7. System.out.println("Elements of given array present on even position: ");
8. //Loop through the array by incrementing value of i by 2
9. //Here, i will start from 1 as first even positioned element is present at position 1.
10. **for** (**int** i = 1; i < arr.length; i = i+2) {
11. System.out.println(arr[i]);
12. }
13. }
14. }

**Output:**

Elements of given array present on even position:

2

4

**7.Java Program to print the elements of an array present on odd position**

Program:

1. **public** **class** OddPosition {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr = **new** **int** [] {1, 2, 3, 4, 5};
5. System.out.println("Elements of given array present on odd position: ");
6. //Loop through the array by incrementing value of i by 2
7. **for** (**int** i = 0; i < arr.length; i = i+2) {
8. System.out.println(arr[i]);
9. }
10. }
11. }

**Output:**

Elements of given array present on odd position:

1

3

5

**8.Java Program to print the largest element in an array .**

1. **public** **class** LargestElement\_array {
2. **public** **static** **void** main(String[] args) {
4. //Initialize array
5. **int** [] arr = **new** **int** [] {25, 11, 7, 75, 56};
6. //Initialize max with first element of array.
7. **int** max = arr[0];
8. //Loop through the array
9. **for** (**int** i = 0; i < arr.length; i++) {
10. //Compare elements of array with max
11. **if**(arr[i] > max)
12. max = arr[i];
13. }
14. System.out.println("Largest element present in given array: " + max);
15. }
16. }

**Output:**

Largest element present in given array: 75

**9.Java Program to print the smallest element in an array .**

1. **public** **class** SmallestElement\_array {
2. **public** **static** **void** main(String[] args) {
4. //Initialize array
5. **int** [] arr = **new** **int** [] {25, 11, 7, 75, 56};
6. //Initialize min with first element of array.
7. **int** min = arr[0];
8. //Loop through the array
9. **for** (**int** i = 0; i < arr.length; i++) {
10. //Compare elements of array with min
11. **if**(arr[i] <min)
12. min = arr[i];
13. }
14. System.out.println("Smallest element present in given array: " + min);
15. }
16. }

**Output:**

Smallest element present in given array: 7

**10.Java Program to print the number of elements present in an array.**

1. **public** **class** CountArray {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr = **new** **int** [] {1, 2, 3, 4, 5};
5. //Number of elements present in an array can be found using the length
6. System.out.println("Number of elements present in given array: " + arr.length);
7. }
8. }

**Output:**

**00:00/05:45**

10

39.9M

908

OOPs Concepts in Java

Number of elements present in given array: 5

**11.Java Program to print the sum of all the items of the array .**

1. **public** **class** SumOfArray {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr = **new** **int** [] {1, 2, 3, 4, 5};
5. **int** sum = 0;
6. //Loop through the array to calculate sum of elements
7. **for** (**int** i = 0; i < arr.length; i++) {
8. sum = sum + arr[i];
9. }
10. System.out.println("Sum of all the elements of an array: " + sum);
11. }
12. }

**Output:**

Sum of all the elements of an array: 15

**12. Java Program to sort the elements of an array in ascending order**

1. **public** **class** SortAsc {
2. **public** **static** **void** main(String[] args) {
4. //Initialize array
5. **int** [] arr = **new** **int** [] {5, 2, 8, 7, 1};
6. **int** temp = 0;
8. //Displaying elements of original array
9. System.out.println("Elements of original array: ");
10. **for** (**int** i = 0; i < arr.length; i++) {
11. System.out.print(arr[i] + " ");
12. }
14. //Sort the array in ascending order
15. **for** (**int** i = 0; i < arr.length; i++) {
16. **for** (**int** j = i+1; j < arr.length; j++) {
17. **if**(arr[i] > arr[j]) {
18. temp = arr[i];
19. arr[i] = arr[j];
20. arr[j] = temp;
21. }
22. }
23. }
25. System.out.println();
27. //Displaying elements of array after sorting
28. System.out.println("Elements of array sorted in ascending order: ");
29. **for** (**int** i = 0; i < arr.length; i++) {
30. System.out.print(arr[i] + " ");
31. }
32. }
33. }

**Output:**

Elements of original array:

5 2 8 7 1

Elements of array sorted in ascending order:

* 1. 2 5 7 8

**13.Java Program to sort the elements of an array in descending order**

1. **public** **class** SortDsc {
2. **public** **static** **void** main(String[] args) {
3. //Initialize array
4. **int** [] arr = **new** **int** [] {5, 2, 8, 7, 1};
5. **int** temp = 0;
7. //Displaying elements of original array
8. System.out.println("Elements of original array: ");
9. **for** (**int** i = 0; i < arr.length; i++) {
10. System.out.print(arr[i] + " ");
11. }
13. //Sort the array in descending order
14. **for** (**int** i = 0; i < arr.length; i++) {
15. **for** (**int** j = i+1; j < arr.length; j++) {
16. **if**(arr[i] < arr[j]) {
17. temp = arr[i];
18. arr[i] = arr[j];
19. arr[j] = temp;
20. }
21. }
22. }
24. System.out.println();
26. //Displaying elements of array after sorting
27. System.out.println("Elements of array sorted in descending order: ");
28. **for** (**int** i = 0; i < arr.length; i++) {
29. System.out.print(arr[i] + " ");
30. }
31. }
32. }

**Output:**

Elements of original array:

5 2 8 7 1

Elements of array sorted in descending order:

* 1. 7 5 2 1

**13.Java Program to Find 3rd Largest Number in an array .**

1. **public** **class** ThirdLargestInArrayExample{
2. **public** **static** **int** getThirdLargest(**int**[] a, **int** total){
3. **int** temp;
4. **for** (**int** i = 0; i < total; i++)
5. {
6. **for** (**int** j = i + 1; j < total; j++)
7. {
8. **if** (a[i] > a[j])
9. {
10. temp = a[i];
11. a[i] = a[j];
12. a[j] = temp;
13. }
14. }
15. }
16. **return** a[total-3];
17. }
18. **public** **static** **void** main(String args[]){
19. **int** a[]={1,2,5,6,3,2};
20. **int** b[]={44,66,99,77,33,22,55};
21. System.out.println("Third Largest: "+getThirdLargest(a,6));
22. System.out.println("Third Largest: "+getThirdLargest(b,7));
23. }}

Output:

Third Largest:3

Third Largest:66

**15. Java Program to Find 2nd Largest Number in an array .**

1. **public** **class** SecondLargestInArrayExample{
2. **public** **static** **int** getSecondLargest(**int**[] a, **int** total){
3. **int** temp;
4. **for** (**int** i = 0; i < total; i++)
5. {
6. **for** (**int** j = i + 1; j < total; j++)
7. {
8. **if** (a[i] > a[j])
9. {
10. temp = a[i];
11. a[i] = a[j];
12. a[j] = temp;
13. }
14. }
15. }
16. **return** a[total-2];
17. }
18. **public** **static** **void** main(String args[]){
19. **int** a[]={1,2,5,6,3,2};
20. **int** b[]={44,66,99,77,33,22,55};
21. System.out.println("Second Largest: "+getSecondLargest(a,6));
22. System.out.println("Second Largest: "+getSecondLargest(b,7));
23. }}

Output:

Second Largest: 5

Second Largest: 77

**16. Java Program to Find Largest Number in an array .**

1. **ublic** **class** LargestInArrayExample{
2. **public** **static** **int** getLargest(**int**[] a, **int** total){
3. **int** temp;
4. **for** (**int** i = 0; i < total; i++)
5. {
6. **for** (**int** j = i + 1; j < total; j++)
7. {
8. **if** (a[i] > a[j])
9. {
10. temp = a[i];
11. a[i] = a[j];
12. a[j] = temp;
13. }
14. }
15. }
16. **return** a[total-1];
17. }
18. **public** **static** **void** main(String args[]){
19. **int** a[]={1,2,5,6,3,2};
20. **int** b[]={44,66,99,77,33,22,55};
21. System.out.println("Largest: "+getLargest(a,6));
22. System.out.println("Largest: "+getLargest(b,7));
23. }}

Output:

Largest: 6

Largest: 99

**17. Java to Program Find 2nd Smallest Number in an array .**

1. **public** **class** SecondSmallestInArrayExample{
2. **public** **static** **int** getSecondSmallest(**int**[] a, **int** total){
3. **int** temp;
4. **for** (**int** i = 0; i < total; i++)
5. {
6. **for** (**int** j = i + 1; j < total; j++)
7. {
8. **if** (a[i] > a[j])
9. {
10. temp = a[i];
11. a[i] = a[j];
12. a[j] = temp;
13. }
14. }
15. }
16. **return** a[1];//2nd element because index starts from 0
17. }
18. **public** **static** **void** main(String args[]){
19. **int** a[]={1,2,5,6,3,2};
20. **int** b[]={44,66,99,77,33,22,55};
21. System.out.println("Second smallest: "+getSecondSmallest(a,6));
22. System.out.println("Second smallest: "+getSecondSmallest(b,7));
23. }}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=SecondSmallestInArrayExample)

Output:

Second smallest: 2

Second smallest: 33

**18. Java Program to Find Smallest Number in an array**

1. **public** **class** SmallestInArrayExample{
2. **public** **static** **int** getSmallest(**int**[] a, **int** total){
3. **int** temp;
4. **for** (**int** i = 0; i < total; i++)
5. {
6. **for** (**int** j = i + 1; j < total; j++)
7. {
8. **if** (a[i] > a[j])
9. {
10. temp = a[i];
11. a[i] = a[j];
12. a[j] = temp;
13. }
14. }
15. }
16. **return** a[0];
17. }
18. **public** **static** **void** main(String args[]){
19. **int** a[]={1,2,5,6,3,2};
20. **int** b[]={44,66,99,77,33,22,55};
21. System.out.println("Smallest: "+getSmallest(a,6));
22. System.out.println("Smallest: "+getSmallest(b,7));
23. }}

Output:

Smallest: 1

Smallest: 22

**19. Java Program to Remove Duplicate Element in an array .**

## Remove Duplicate Element in Array using Temporary Array

1. **public** **class** RemoveDuplicateInArrayExample{
2. **public** **static** **int** removeDuplicateElements(**int** arr[], **int** n){
3. **if** (n==0 || n==1){
4. **return** n;
5. }
6. **int**[] temp = **new** **int**[n];
7. **int** j = 0;
8. **for** (**int** i=0; i<n-1; i++){
9. **if** (arr[i] != arr[i+1]){
10. temp[j++] = arr[i];
11. }
12. }
13. temp[j++] = arr[n-1];
14. // Changing original array
15. **for** (**int** i=0; i<j; i++){
16. arr[i] = temp[i];
17. }
18. **return** j;
19. }
21. **public** **static** **void** main (String[] args) {
22. **int** arr[] = {10,20,20,30,30,40,50,50};
23. **int** length = arr.length;
24. length = removeDuplicateElements(arr, length);
25. //printing array elements
26. **for** (**int** i=0; i<length; i++)
27. System.out.print(arr[i]+" ");
28. }
29. }

Output:

10 20 30 40 50

**20. Java Program to Print Odd and Even Numbers from an array .**

1. **public** **class** OddEvenInArrayExample{
2. **public** **static** **void** main(String args[]){
3. **int** a[]={1,2,5,6,3,2};
4. System.out.println("Odd Numbers:");
5. **for**(**int** i=0;i<a.length;i++){
6. **if**(a[i]%2!=0){
7. System.out.println(a[i]);
8. }
9. }
10. System.out.println("Even Numbers:");
11. **for**(**int** i=0;i<a.length;i++){
12. **if**(a[i]%2==0){
13. System.out.println(a[i]);
14. }
15. }
16. }}

Output:

Odd Numbers:

1

5

3

Even Numbers:

2

6

2

**21.How to Sort an Array in Java**

1. **import** java.util.Arrays;
2. **public** **class** SortArrayExample1
3. {
4. **public** **static** **void** main(String[] args)
5. {
6. //defining an array of integer type
7. **int** [] array = **new** **int** [] {90, 23, 5, 109, 12, 22, 67, 34};
8. //invoking sort() method of the Arrays class
9. Arrays.sort(array);
10. System.out.println("Elements of array sorted in ascending order: ");
11. //prints array using the for loop
12. **for** (**int** i = 0; i < array.length; i++)
13. {
14. System.out.println(array[i]);
15. }
16. }
17. }

**Output:**

Array elements in ascending order:

5

12

22

23

34

67

90

109

**Java Number Programs**

**1.How to Reverse a Number in Java**.

1. **public** **class** ReverseNumberExample1
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** number = 987654, reverse = 0;
6. **while**(number != 0)
7. {
8. **int** remainder = number % 10;
9. reverse = reverse \* 10 + remainder;
10. number = number/10;
11. }
12. System.out.println("The reverse of the given number is: " + reverse);
13. }  }

**Output**

The reverse of the given number is: 456789

**2. Java Program to Display Even Numbers From 1 to 100 .**

1. **public** **class** DisplayEvenNumbersExample1
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **int** number=100;
6. System.out.print("List of even numbers from 1 to "+number+": ");
7. **for** (**int** i=1; i<=number; i++)
8. {
9. //logic to check if the number is even or not
10. //if i%2 is equal to zero, the number is even
11. **if** (i%2==0)
12. {
13. System.out.print(i + " ");
14. }
15. }
16. }
17. }

**Output:**

2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100

**3. Java Program to Display Odd Numbers From 1 to 100**

1. **public** **class** DisplayOddNumbersExample1
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **int** number=100;
6. System.out.print("List of odd numbers from 1 to "+number+": ");
7. **for** (**int** i=1; i<=number; i++)
8. {
9. //logic to check if the number is odd or not
10. //if i%2 is not equal to zero, the number is odd
11. **if** (i%2!=0)
12. {
13. System.out.print(i + " ");  }  }  }  }

List of odd numbers from 1 to 100: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99

**4. Java Program to Prime Number Program in Java .**

1. **public** **class** PrimeExample{
2. **public** **static** **void** main(String args[]){
3. **int** i,m=0,flag=0;
4. **int** n=3;//it is the number to be checked
5. m=n/2;
6. **if**(n==0||n==1){
7. System.out.println(n+" is not prime number");
8. }**else**{
9. **for**(i=2;i<=m;i++){
10. **if**(n%i==0){
11. System.out.println(n+" is not prime number");
12. flag=1;
13. **break**;
14. }
15. }
16. **if**(flag==0)  { System.out.println(n+" is prime number"); }
17. }//end of else
18. }
19. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=PrimeExample)

Output:

3 is prime number

List of even numbers from 1 to 100: 2 4 6 8 10 12 14 16 18 20 22 24 26 28

**Basic Java Programs**

**1. Java Program to find Fibonacci Series**

1. **class** FibonacciExample1{
2. **public** **static** **void** main(String args[])
3. {
4. **int** n1=0,n2=1,n3,i,count=10;
5. System.out.print(n1+" "+n2);//printing 0 and 1
7. **for**(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
8. {
9. n3=n1+n2;
10. System.out.print(" "+n3);
11. n1=n2;
12. n2=n3;
13. }
15. }}

Output:57

Competitive questions on Structures

0 1 1 2 3 5 8 13 21 34

**2. Java Program to check number is Palindrome or not .**

1. **class** PalindromeExample{
2. **public** **static** **void** main(String args[]){
3. **int** r,sum=0,temp;
4. **int** n=454;//It is the number variable to be checked for palindrome
6. temp=n;
7. **while**(n>0){
8. r=n%10;  //getting remainder
9. sum=(sum\*10)+r;
10. n=n/10;
11. }
12. **if**(temp==sum)
13. System.out.println("palindrome number ");
14. **else**
15. System.out.println("not palindrome");
16. }
17. }

Output:

palindrome number

**3. Java Program to find Factorial of number**

1. **class** FactorialExample{
2. **public** **static** **void** main(String args[]){
3. **int** i,fact=1;
4. **int** number=5;//It is the number to calculate factorial
5. **for**(i=1;i<=number;i++){
6. fact=fact\*i;
7. }
8. System.out.println("Factorial of "+number+" is: "+fact);
9. }
10. }

Output:

Factorial of 5 is: 120

**4. How to Generate Random Number in Java .**

1. **import** java.lang.Math;
2. **public** **class** RandomNumberExample1
3. {
4. **public** **static** **void** main(String args[])
5. {
6. // Generating random numbers
7. System.out.println("1st Random Number: " + Math.random());
8. System.out.println("2nd Random Number: " + Math.random());
9. System.out.println("3rd Random Number: " + Math.random());
10. System.out.println("4th Random Number: " + Math.random());
11. }
12. }

**5. How to Print Pattern in Java**

The execution of the code step by step, for **n=4** (the number of rows we want to print).

**Iteration 1:**

For i=0, 0<4 (true)

For j=0, j<=0 (true)

The first **print** statement prints a star at the first row and the second **println** statement prints the spaces and throws the cursor at the next line.

Now the value of i and j is increased to 1.

**Iteration 2:**

For i=1, 1<4 (true)

For j=1, 1<=1 (true)

The first **print** statement prints two stars at the second row and the second **println** statement prints the spaces and throws the cursor at the next line.

1. \*
2. \* \*

Now the value of i and j is increased to 2.

**Iteration 3:**

For i=2, 2<4 (true)

For j=2, 2<=2 (true)

The first **print** statement prints three stars at the third row and the second **println** statement prints the spaces and throws the cursor at the next line.

1. \*
2. \* \*
3. \* \* \*

Now the value of i and j is increased to 3.

**Iteration 4:**

For i=3, 3<4 (true)

For j=3, 3<=3 (true)

The first **print** statement prints four stars at the fourth row and the second **println** statement prints the spaces and throws the cursor at the next line.

1. \*
2. \* \*
3. \* \* \*
4. \* \* \* \*

Now the value of i and j is increased to 4.

For i=4, 4<4 (false)

The execution of the program will terminate when the value of i will be equal to the number of rows.

**6. How to Compare Two Objects in Java**

## Java equals() Method

The **equals()** method of the Object class compare the equality of two objects. The two objects will be equal if they share the same memory address.

**Syntax:**HTML Tutorial

**public** **boolean** equals(Object obj)

The method parses a reference object as a parameter. It returns true if the objects are equal, else returns false.

### Example of equals() method

1. **public** **class** ObjectComparisonExample
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. //creating constructor of the Double class
6. Double x = **new** Double(123.45555);
7. //creating constructor of the Long class
8. Long y = **new** Long(9887544);
9. //invoking the equals() method
10. System.out.println("Objects are not equal, hence it returns " + x.equals(y));
11. System.out.println("Objects are equal, hence it returns " + x.equals(123.45555));
12. }
13. }

**Output:**

Objects are not equal, hence it returns false

Objects are equal, hence it returns true

**7. How to Create Object in Java**

The **object** is a basic building block of an [OOPs](https://www.javatpoint.com/java-oops-concepts) language. In **Java**, we cannot execute any program without creating an **object**. There is various way to **create an**[**object in Java**](https://www.javatpoint.com/object-and-class-in-java) that we will discuss in this section, and also learn **how to create an object in Java.**

[Java](https://www.javatpoint.com/java-tutorial) provides five ways to create an object.

* Using **new** Keyword
* Using **clone()** method
* Using **newInstance()** method of the **Class** class
* Using **newInstance()** method of the **Constructor** class
* Using **Deserialization**

## Using new Keyword

Using the **new** keyword is the most popular way to create an object or instance of the class. When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created **object** and also returns the **reference** of that object to that memory. The new keyword is also used to create an array. The syntax for creating an object is:

1. ClassName object = **new** ClassName();

Let's create a program that uses new keyword to create an object.

Accessing Elements of Union in C in Hindi

Keep Watching

**CreateObjectExample1.java**

1. **public** **class** CreateObjectExample1
2. {
3. **void** show()
4. {
5. System.out.println("Welcome to javaTpoint");
6. }
7. **public** **static** **void** main(String[] args)
8. {
9. //creating an object using new keyword
10. CreateObjectExample1 obj = **new** CreateObjectExample1();
11. //invoking method using the object
12. obj.show();
13. }
14. }

**Output:**

Welcome to javaTpoint

By using the new keyword, we can also invoke the constructor (default or parametrized) of the class.

**CreateObjectExample2.java**

1. **public** **class** CreateObjectExample2
2. {
3. //constructor of the class
4. CreateObjectExample2()
5. {
6. System.out.println("Welcome to javaTpoint");
7. }
8. **public** **static** **void** main(String[] args)
9. {
10. //creating an object using new keyword
11. CreateObjectExample2 obj = **new** CreateObjectExample2();
12. }
13. }

**Output:**

Welcome to javaTpoint

## Using clone() Method

The **clone()** method is the method of **Object** class. It creates a copy of an object and returns the same copy. The [JVM](https://www.javatpoint.com/jvm-java-virtual-machine)

creates a new object when the clone() method is invoked. It copies all the content of the previously created object into new one object. Note that it does not call any constructor. We must implement the **Cloneable** interface while using the clone() method. The method throws **CloneNotSupportedException** exception if the object's class does not support the Cloneable interface. The subclasses that override the clone() method can throw an exception if an instance cannot be cloned.

#### Note: The method creates a copy of the object not a new object.

**Syntax:**

1. **protected** Object clone() **throws** CloneNotSupportedException

We use the following statement to create a new object.

1. ClassName newobject = (ClassName) oldobject.clone();

**CreateObjectExample3.java**

1. **public** **class** CreateObjectExample3 **implements** Cloneable
2. {
3. @Override
4. **protected** Object clone() **throws** CloneNotSupportedException
5. {
6. //invokes the clone() method of the super class
7. **return** **super**.clone();
8. }
9. String str = "New Object Created";
10. **public** **static** **void** main(String[] args)
11. {
12. //creating an object of the class
13. CreateObjectExample3 obj1 = **new** CreateObjectExample3();
14. //try catch block to catch the exception thrown by the method
15. **try**
16. {
17. //creating a new object of the obj1 suing the clone() method
18. CreateObjectExample3 obj2 = (CreateObjectExample3) obj1.clone();
19. System.out.println(obj2.str);
20. }
21. **catch** (CloneNotSupportedException e)
22. {
23. e.printStackTrace();
24. }
25. }
26. }

**Output:**

New Object Created

## Using newInstance() Method of Class class

The **newInstance()** method of the Class class is also used to create an object. It calls the default constructor to create the object. It returns a newly created instance of the class represented by the object. It internally uses the newInstance() method of the Constructor class.

**Syntax:**

1. **public** T newInstance() **throws** InstantiationException, IllegalAccessException

It throws the **IllegalAccessException,** **InstantiationException, ExceptionInInitializerError** exceptions.

We can create an object in the following ways:

1. ClassName object = ClassName.**class**.newInstance();

**Or**

1. ClassName object = (ClassName) Class.forName("fully qualified name of the class").newInstance();

In the above statement, **forName()** is a static method of Class class. It parses a parameter **className** of type String. It returns the object for the class with the fully qualified name. It loads the class but does not create any object. It throws **ClassNotFoundException** if the class cannot be loaded and **LinkageError** if the linkage fails.

To create the object, we use the **newInstance()** method of the Class class. It works only when we know the name of the class and the class has a public default constructor.

In the following program, we have creates a new object using the newInstance() method.

**CreateObjectExample4.java**

1. **public** **class** CreateObjectExample4
2. {
3. **void** show()
4. {
5. System.out.println("A new object created.");
6. }
7. **public** **static** **void** main(String[] args)
8. {
9. **try**
10. {
11. //creating an instance of Class class
12. Class cls = Class.forName("CreateObjectExample4");
13. //creates an instance of the class using the newInstance() method
14. CreateObjectExample4 obj = (CreateObjectExample4) cls.newInstance();
15. //invoking the show() method
16. obj.show();
17. }
18. **catch** (ClassNotFoundException e)
19. {
20. e.printStackTrace();
21. }
22. **catch** (InstantiationException e)
23. {
24. e.printStackTrace();
25. }
26. **catch** (IllegalAccessException e)
27. {
28. e.printStackTrace();
29. }
30. }
31. }

**Output:**

A new object created.

## Using newInstance() Method of Constructor class

It is similar to the **newInstance()** method of the **Class** class. It is known as a reflective way to create objects. The method is defined in the **Constructor** class which is the class of java.lang.reflect package. We can also call the parameterized constructor and private constructor by using the **newInstance()** method. It is widely preferred in comparison to [newInstance() method](https://www.javatpoint.com/new-instance()-method)

of the Class class.

**Syntax:**

1. **public** T newInstance(Object... initargs) **throws** InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException

The method parses an array of Objects as an argument. The values of primitive types wrapped in a wrapper Object of the appropriate type. It returns a new object created by calling the constructor. It throws **IllegalAccessException, IllegalArgumentException, InstantiationException, InvocationTargetException, ExceptionInInitializerError Exceptions**.

We can create an object in the following way:

1. Constructor<Employee> constructor = Employee.**class**.getConstructor();
2. Employee emp3 = constructor.newInstance();

Let's create a program that creates an object using the newInstance() method.

**CreateObjectExample5.java**

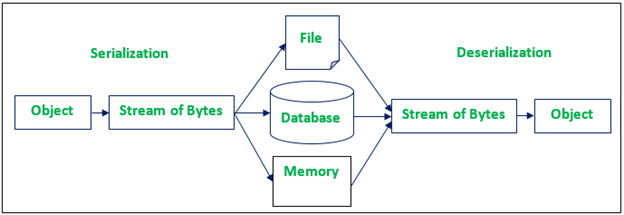
1. **import** java.lang.reflect.\*;
2. **public** **class** CreateObjectExample5
3. {
4. **private** String str;
5. CreateObjectExample5()
6. {
7. }
8. **public** **void** setName(String str)
9. {
10. **this**.str = str;
11. }
12. **public** **static** **void** main(String[] args)
13. {
14. **try**
15. {
16. Constructor<CreateObjectExample5> constructor = CreateObjectExample5.**class**.getDeclaredConstructor();
17. CreateObjectExample5 r = constructor.newInstance();
18. r.setName("JavaTpoint");
19. System.out.println(r.str);
20. }
21. **catch** (Exception e)
22. {
23. e.printStackTrace();
24. }
25. }
26. }

**Output:**

JavaTpoint

## Using Deserialization

In Java, **serialization** is the process of converting an object into a sequence of byte-stream. The reverse process (byte-stream to object) of serialization is called **deserialization**. The JVM creates a new object when we serialize or deserialize an object. It does not use constructor to create an object. While using deserialization, the **Serializable** interface (marker interface) must be implemented in the class.



**Serialization:** The **writeObject()** method of the **ObjectOutputStream** class is used to serialize an object. It sends the object to the output stream.

**Syntax:**

1. **public** **final** **void** writeObject(object x) **throws** IOException

**Deserialization:** The method **readObject()** of **ObjectInputStream** class is used to deserialize an object. It references objects out of a stream.

**Syntax:**

1. **public** **final** Object readObject() **throws** IOException,ClassNotFoundException

#### Note: Make the filed static or transient if we do not want to include a field as a part of the object. It will not include in the serialization process.

Let's understand the serialization and deserialization through a program.

**Employee.java**

1. **import** java.io.Serializable;
2. **public** **class** Employee **implements** Serializable
3. {
4. **int** empid;
5. String empname;
6. **public** Empoyee(**int** empid, String empname)
7. {
8. **this**.empid = empid;
9. **this**.empname = empname;
10. }
11. }

We have created a class named **Employee** whose object is to be serialized and deserialized.

### Serialization of Java Object:

In the following program, we have serialized an object of Employee class by using the **writeObject()** method of the ObjectOutputStream class. The state of the object is saved in the **employee.txt** file.

**SerializationExample.java**

1. **import** java.io.\*;
2. **class** SerializationExample
3. {
4. **public** **static** **void** main(String args[])
5. {
6. Try
7. {
8. //Creating the object
9. Employee emp = **new** Employee(198054,"Andrew");
10. //Creates a stream and writes the object
11. FileOutputStream fout=**new** FileOutputStream("employee.txt");
12. ObjectOutputStream out=**new** ObjectOutputStream(employeeout);
13. out.writeObject(emp);
14. out.flush();
15. //closes the output stream
16. out.close();
17. System.out.println("Successfully Created");
18. }
19. **catch**(Exception e)
20. {
21. System.out.println(e);
22. }
23. }
24. }

**Output:**

Successfully Created

### Deserialization of Java Object:

In the following program, we going to deserialize an object that we have serialized in the above program.

**DeserializationExample.java**

1. **import** java.io.\*;
2. **class** DeserializationExample
3. {
4. **public** **static** **void** main(String args[])
5. {
6. **try**
7. {
8. //Creating a stream to read the object
9. ObjectInputStream in=**new** ObjectInputStream(**new** FileInputStream("employee.txt"));
10. Employee e=(Employee)in.readObject();
11. //prints the data of the serialized object
12. System.out.println(e.empid+" "+e.empname);
13. //closing the input stream
14. in.close();
15. }
16. **catch**(Exception e)
17. {
18. System.out.println(e);
19. }
20. }
21. }

**Output:**

198054 Andrew

In the above five methods, we have noticed that the **new** keyword and both **newInstance()** methods use the constructor to create objects, while the rest two methods do not use the constructor.