SUMMER TRAINING REPORT

On

Airline Reservation System

# A Project Report submitted in partial fulfilment of

the requirement for the award of

## Bachelor of Engineering

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted By:**

**Parina**

**(Roll no. CO17343)**

Under the supervision of

The Society for Promotion of IT in Chandigarh (SPIC)

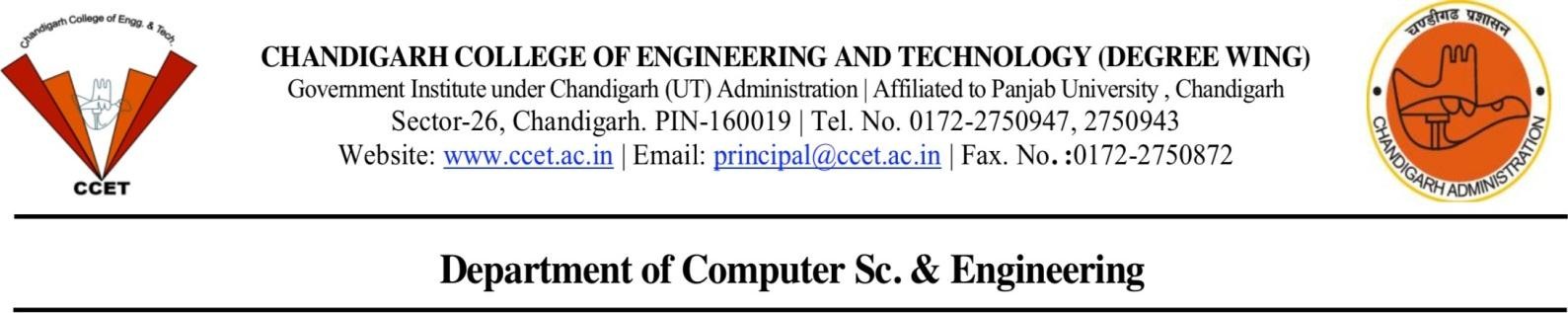
****

## Chandigarh College of Engineering And Technology

### (DEGREE WING)

Government Institute under Chandigarh (UT) Administration , Affiliated to Punjab University

Chandigarh,Sec-26



CONTENTS

Contents

Student declaration………………………………………………………………………..1

Certificate by the guide…………………………………………………………………....2

Acknowledgement…………………………………………………………………………3

Abstract…………………………………………………………………………………….4

[INTRODUCTION](#page8)

[1.1](#page8) JAVA……………………………………………………………………………………

1.2 HISTORY OF JAVA……………………………………………………………………………………

1.3 JAVA FEATURES……………………………………………………………………………

[OPERATORS](#page11)

COLLECTIONS IN JAVA………………………………………………………………

FUNCTIONS IN JAVA…………………………………………………………………

JAVA MODULES……………………………………………………………………….

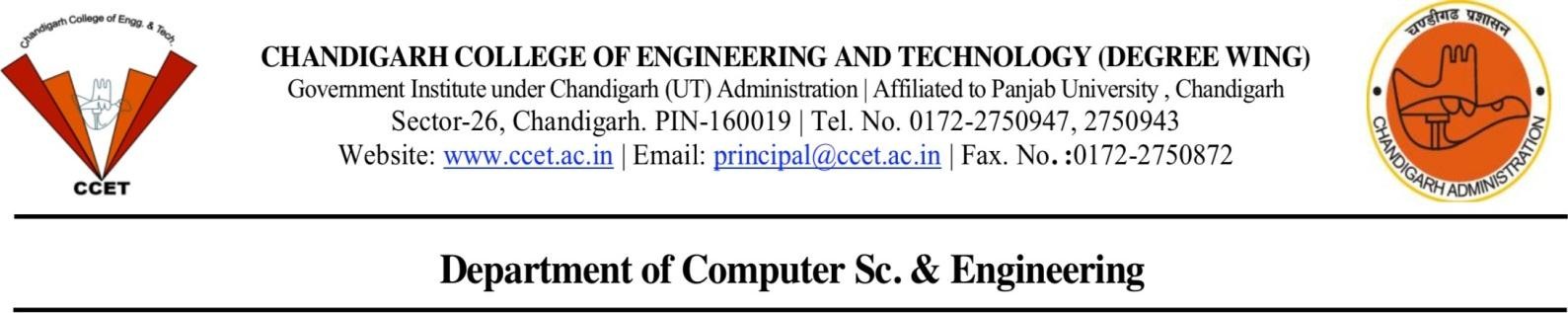
JAVA I/O FILES………………………………………………………………………

JAVA OBJECT ORIENTED…………………………………………………………..

JAVA EXCEPTION HANDLING………………………………………………………

JAVA DATABASE MYSQL ACCESS……………………………………………

OUTPUTS …………………………………………………………………………………..

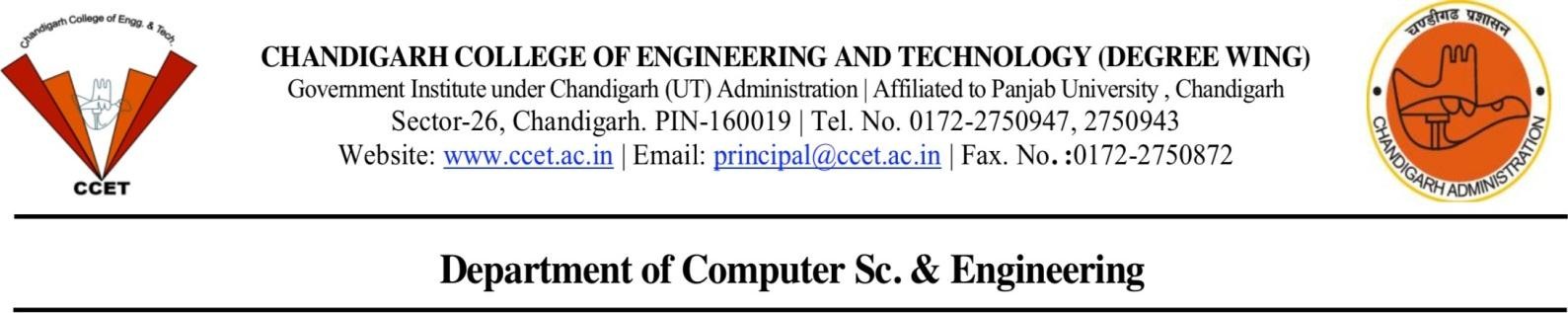


**STUDENT DECLARTION**

This is declare to that the work presented in this report entitled Airline Reservation System, in the fulfillment of the requirement for the award of the Bachelor of Engineering in Computer Sciences & Engineering, submitted in CSE Department , Chandigarh College of Engineering & Technology (Degree Wing) affiliated to Punjab University Chandigarh is an authentic record of my/our work carried out during my degree under the guidance of Richy Longia .The work reported in this has not submitted by me for award of any other degree or diploma.

Date : 17/12/19 Parina

Place: CCET CO17343



**CERTIFICATE**

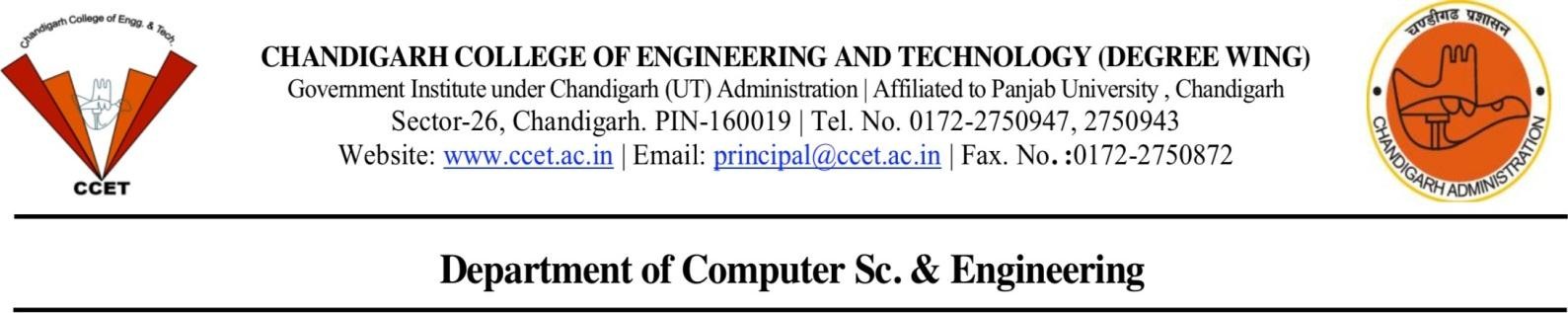
This is to certify that the Project work entitled “**Airline Reservation System** submitted by **Parina , roll no.:CO17343** fulfillment for the requirements of the award of Bachelor of Engineering Degree in Computer Science & Engineering at Chandigarh College of Engineering and Technology (Degree Wing), Chandigarh is an authentic work carried out by him/her under my supervision and guidance.

To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree .

Date : 17/12/19

Place : CCET Dept of CSE

CCET(Degree Wing)

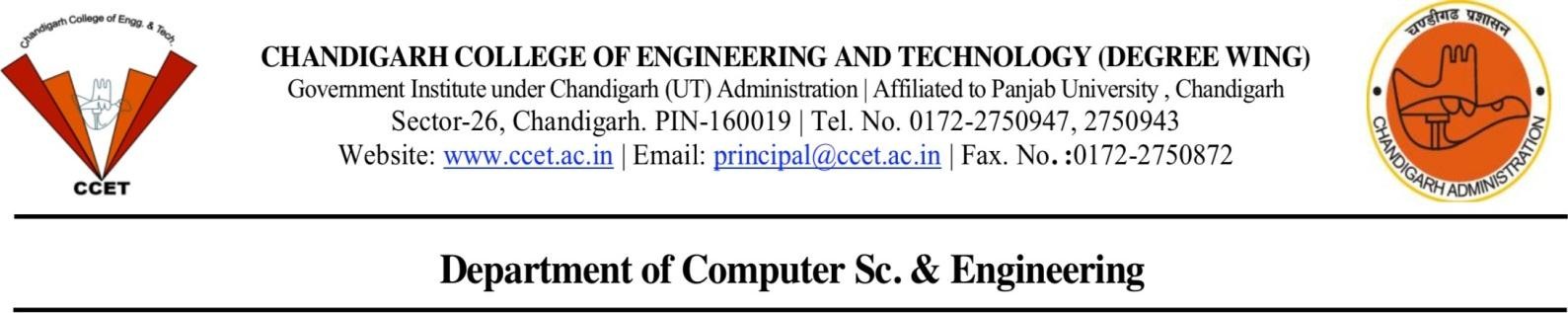


**ACKNOWLEDGEMENT**

“Any serious and lasting achievement or success, one can never achieve without the help, guidance and co-operation of so many people involved in the work.

I would like to express deep gratitude to Dr. Sunil K. Singh, Head of Department (Computer Science & Engineering), submitted in CSE Department, Chandigarh College of Engineering &Technology(Degree wing) affiliated to Punjab University, Chandigarh without whose permission the training would not be possible. I would also like to thank Dr. Ankit Gupta, Training & Placement Officer, CSE. Department, who recommended me for this training.

I have tried my best to keep report simple yet technically correct. I hope I succeed in my attempt.



**ABSTRACT**

The objective of a practical training is to learn something about industries practically and to be familiar with a working style of a technical worker to adjust simply according to industrial environment . This report deals with the equipments their relation and their general operating principle.

**Java** is a [general-purpose](https://en.wikipedia.org/wiki/General-purpose_language) [programming language](https://en.wikipedia.org/wiki/Programming_language) that is [class-based](https://en.wikipedia.org/wiki/Class-based_programming), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), and designed to have as few implementation [dependencies](https://en.wikipedia.org/wiki/Dependency_(computer_science)) as possible. It is intended to let [application developers](https://en.wikipedia.org/wiki/Application_developer) write once, run anywhere (WORA), meaning that [compiled](https://en.wikipedia.org/wiki/Compiler) Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to [bytecode](https://en.wikipedia.org/wiki/Java_bytecode) that can run on any [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) regardless of the underlying [computer architecture](https://en.wikipedia.org/wiki/Computer_architecture). The [syntax](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) of [Java](https://en.wikipedia.org/wiki/Java_(software_platform)) is similar to [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B), but it has fewer [low-level](https://en.wikipedia.org/wiki/Low-level_programming_language) facilities than either of them. As of 2019, Java was one of the most [popular programming languages in use](https://en.wikipedia.org/wiki/Measuring_programming_language_popularity) according to [GitHub](https://en.wikipedia.org/wiki/GitHub), particularly for [client-server](https://en.wikipedia.org/wiki/Client%E2%80%93server) [web applications](https://en.wikipedia.org/wiki/Web_applications), with a reported 9 million developers.

**INTRODUCTION**

* 1. **JAVA**

Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than **3 billion** devices run Java.

It is used for:

* Mobile applications (specially Android apps)
* Desktop applications
* Web applications
* Web servers and application servers
* Games
* Database connection
* And much, much more!

## Why Use Java?

* Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
* It is one of the most popular programming language in the world
* It is easy to learn and simple to use
* It is open-source and free
* It is secure, fast and powerful
* It has a huge community support (tens of millions of developers)
* Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
* As Java is close to [C++](https://www.w3schools.com/cpp/default.asp) and [C#](https://www.w3schools.com/cs/default.asp), it makes it easy for programmers to switch to Java or vice versa.

**1.2 HISTORY OF JAVA**

**The history of Java** is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as **Green Team**), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given significant points that describe the history of Java.

1) [**James Gosling**](https://www.javatpoint.com/james-gosling-father-of-java), **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Initially designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

7) **Why had they chosen java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

8) Java is an island of Indonesia where the first coffee was produced (called java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having coffee near his office.

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at [Sun Microsystems](https://www.javatpoint.com/sun-microsystems) (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in(January 23, 1996). After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds the new features in Java.

**1.3 JAVA FEATURES**

A list of most important features of Java language is given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

**OPERATORS**



## 2.1 Java Operator Precedence

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Category** | **Precedence** |
| Unary | postfix | *expr*++ *expr*-- |
| prefix | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

Lets take a look at them in detail.

1. **Arithmetic Operators:** They are used to perform simple arithmetic operations on primitive data types.
   * **\* :** Multiplication
   * **/ :** Division
   * **% :** Modulo
   * **+ :** Addition
   * **– :** Subtraction

**2.Unary Operators:** Unary operators need only one operand. They are used to increment, decrement or negate a value.

* **– :Unary minus**, used for negating the values.
* **+ :Unary plus**, used for giving positive values. Only used when deliberately converting a negative value to positive.
* **++ :Increment operator**, used for incrementing the value by 1. There are two varieties of increment operator.
  + **Post-Increment :** Value is first used for computing the result and then incremented.
  + **Pre-Increment :** Value is incremented first and then result is computed.
* **— : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operator.
  + **Post-decrement :** Value is first used for computing the result and then decremented.
  + **Pre-Decrement :** Value is decremented first and then result is computed.
* **! : Logical not operator**, used for inverting a boolean value.

3. **Assignment Operator : ‘=’** Assignment operator is used to assign a value to any variable. It has a right to left associativity, i.e value given on right hand side of operator is assigned to the variable on the left and therefore right hand side value must be declared before using it or should be a constant.  
General format of assignment operator is,

variable **=** value;

In many cases assignment operator can be combined with other operators to build a shorter version of statement called **Compound Statement**. For example, instead of a **=** a+5, we can write a **+=** 5.

* **+=**, for adding left operand with right operand and then assigning it to variable on the left.
* **-=**, for subtracting left operand with right operand and then assigning it to variable on the left.
* **\*=**, for multiplying left operand with right operand and then assigning it to variable on the left.
* **/=**, for dividing left operand with right operand and then assigning it to variable on the left.
* **%=**, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

int a = 5;

a += 5; //a = a+5;

4. **Relational Operators :** These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are extensively used in looping statements as well as conditional if else statements. General format is,

variable **relation\_operator** value

Some of the relational operators are-

* **==, Equal to :** returns true of left hand side is equal to right hand side.
* **!=, Not Equal to :** returns true of left hand side is not equal to right hand side.
* **<, less than :** returns true of left hand side is less than right hand side.
* **<=, less than or equal to :** returns true of left hand side is less than or equal to right hand side.
* **>, Greater than :** returns true of left hand side is greater than right hand side.
* **>=, Greater than or equal to:** returns true of left hand side is greater than or equal to right hand side.

5. **Logical Operators :** These operators are used to perform “logical AND” and “logical OR” operation, i.e. the function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e. it has a short-circuiting effect. Used extensively to test for several conditions for making a decision.  
Conditional operators are-

* **&&, Logical AND :** returns true when both conditions are true.
* **||, Logical OR :** returns true if at least one condition is true.

6. **Ternary operator :** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary. General format is-

condition **?** if true **:** if false

The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.

7, **Bitwise Operators :** These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.

* **&, Bitwise AND operator:** returns bit by bit AND of input values.
* **|, Bitwise OR operator:** returns bit by bit OR of input values.
* **^, Bitwise XOR operator:** returns bit by bit XOR of input values.
* **~, Bitwise Complement Operator:** This is a unary operator which returns the one’s compliment representation of the input value, i.e. with all bits inversed.

8. **Shift Operators :**These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two. General format-

number **shift\_op** number\_of\_places\_to\_shift;

* **<<, Left shift operator:** shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.
* **>>, Signed Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of initial number. Similar effect as of dividing the number with some power of two.
* **>>>, Unsigned Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0,

**9.** [**instance of operator**](https://www.geeksforgeeks.org/java-instanceof-and-its-applications/) **:** Instance of operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass or an interface. General format-

* object **instance of** class/subclass/interface

# COLLECTIONS IN JAVA

1. [Java Collection Framework](https://www.javatpoint.com/collections-in-java)
2. [Hierarchy of Collection Framework](https://www.javatpoint.com/collections-in-java#collectionhierarchy)
3. [Collection interface](https://www.javatpoint.com/collections-in-java#collectionmethods)
4. [Iterator interface](https://www.javatpoint.com/collections-in-java#collectioniterator)

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

#### What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

#### What is a framework in Java

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

#### What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

### Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the classes and interfaces for the Collection framework.

### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(E e) | It is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection<? extends E> c) | It is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | It is used to delete an element from the collection. |
| 4 | public boolean removeAll(Collection<?> c) | It is used to delete all the elements of the specified collection from the invoking collection. |
| 5 | default boolean removeIf(Predicate<? super E> filter) | It is used to delete all the elements of the collection that satisfy the specified predicate. |
| 6 | public boolean retainAll(Collection<?> c) | It is used to delete all the elements of invoking collection except the specified collection. |
| 7 | public int size() | It returns the total number of elements in the collection. |
| 8 | public void clear() | It removes the total number of elements from the collection. |
| 9 | public boolean contains(Object element) | It is used to search an element. |
| 10 | public boolean containsAll(Collection<?> c) | It is used to search the specified collection in the collection. |
| 11 | public Iterator iterator() | It returns an iterator. |
| 12 | public Object[] toArray() | It converts collection into array. |
| 13 | public <T> T[] toArray(T[] a) | It converts collection into array. Here, the runtime type of the returned array is that of the specified array. |
| 14 | public boolean isEmpty() | It checks if collection is empty. |
| 15 | default Stream<E> parallelStream() | It returns a possibly parallel Stream with the collection as its source. |
| 16 | default Stream<E> stream() | It returns a sequential Stream with the collection as its source. |
| 17 | default Spliterator<E> spliterator() | It generates a Spliterator over the specified elements in the collection. |
| 18 | public boolean equals(Object element) | It matches two collections. |
| 19 | public int hashCode() | It returns the hash code number of the collection. |

### Iterator interface

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in a forward direction only. |

#### Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean hasNext() | It returns true if the iterator has more elements otherwise it returns false. |
| 2 | public Object next() | It returns the element and moves the cursor pointer to the next element. |
| 3 | public void remove() | It removes the last elements returned by the iterator. It is less used. |

## Iterable Interface

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

1. Iterator<T> iterator()

It returns the iterator over the elements of type T.

## Collection Interface

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

## List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

## ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.

## LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

## Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

## Stack

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

## Queue Interface

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Queue interface can be instantiated as:

1. Queue<String> q1 = new PriorityQueue();
2. Queue<String> q2 = new ArrayDeque();

There are various classes that implement the Queue interface, some of them are given below.

## PriorityQueue

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

## Deque Interface

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

Deque can be instantiated as:

1. Deque d = new ArrayDeque();

## ArrayDeque

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

## Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

## HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

## LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

## SortedSet Interface

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

The SortedSet can be instantiated as:

1. SortedSet<data-type> set = new TreeSet();

## TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

FUNCTIONS IN JAVA

In Java, all function definitions must be inside classes. We also call functions methods. Let's look at an example method

public class Main {

public static void foo() {

// Do something here

}

}

foo is a method we defined in class Main. Notice a few things about foo.

* static means this method belongs to the class Main and not to a specific instance of Main. Which means we can call the method from a different class like that Main.foo().
* void means this method doesn't return a value. Methods can return a single value in Java and it has to be defined in the method declaration. However, you can use return by itself to exit the method.
* This method doesn't get any arguments, but of course Java methods can get arguments as we'll see further on.

### Arguments

I always like to say that arguments to Java methods are passed by value, although some might disagree with my choice of words, I find it the best way to explain and understand how it works exactly.

By value means that arguments are copied when the method runs. Let's look at an example.

public void bar(int num1, int num2) {

...

}

Here is a another place in the code, where bar is called

int a = 3;

int b = 5;

bar(a, b);

You can picture in your head that when bar(a, b) is run, it's like in the beginning of bar the following two lines are written:

int num1 = a;

int num2 = b;

And only then the rest of the method is run.

This means that a value get copied to num1 and b value get copied to num2. Changing the values of num1 and num2 will not affect a and b.

If the arguments were objects, the rules remain the same, but it acts a bit differently. Here is a an example:

public void bar2(Student s1, Student s2) {

...

}

And here is how we use it

Student joe = new Student("joe");

Student jack = new Student("jack");

bar2(joe, jack);

Again we can picture the same two lines in the beginning of bar2

Student s1 = joe;

Student s2 = jack;

But when we assign objects, it's a bit different than assigning primitives. s1 and joe are two different references to the same object. s1 == joe is true. This means that running methods on s1 will change the object joe. But if we'll change the value of s1 as a reference, it will not affect the reference joe.

s1.setName("Chuck"); // joe name is now Chuck as well

s1 = new Student("Norris"); // s1 is a new student, different than joe with the name of Norris

// s1 == joe is not true anymore

### Non static methods

Non static methods in Java are used more than static methods. Those methods can only be run on objects and not on the whole class.

Non static methods can access and alter the field of the object.

public class Student {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

Calling the methods will require an object of type Student.

Student s = new Student();

s.setName("Danielle");

String name = s.getName();

Student.setName("Bob"); // Will not work!

Student.getName(); // Will not work!

### Summary

* Every Java method has to be within a class
* Static methods belong to a class while non-static methods belong to objects
* All parameters to functions are passed by value, primitives content is copied, while objects are not copied and some would say 'passed by reference'.

## JAVA MODULES

A Java Module is a mechanism to package up your Java application and Java packages into Java modules. A Java module can specify which of the Java packages it contains that should be visible to other Java modules using this module. A Java module must also specify which other Java modules is requires to do its job. This will be explained in more detail later in this Java modules tutorial.

Java modules is a [new feature in Java 9](http://tutorials.jenkov.com/java/index.html#new-in-java-9) via the Java Platform Module System (JPMS). The Java Platform Module System is also sometimes referred to as Java Jigsaw or Project Jigsaw depending on where you read. Jigsaw was the internally used project name during development. Later Jigsaw changed name to Java Platform Module System.

## Java Module Benefits

The Java Platform Module System brings several benefits to us Java developers. I will list the biggest benefits below.

### Smaller Application Distributables via the Modular Java Platform

As part of Project Jigsaw, all the Java Platform APIs have been split up into separate modules. The benefit of splitting all the Java APIs up into modules is that you can now specify what modules of the Java platform your application requires. Knowing what Java Platform modules your application requires, Java can package up your application including only the Java Platform modules that your application actually uses.

Before Java 9 and the Java Platform Module System you would have had to package all of the Java Platform APIs with your Java application because there was no official way of reliably checking what classes your Java application used. Since the Java Platform APIs have grown quite large over the years, your application would get a large amount of Java classes included in its distribution, many of which your application would probably not be using.

The unused classes makes your application distributable bigger than it needs to be. This can be a problem on small devices like mobile phones, Raspberry Pis etc. With the Java Platform Module System you can now package your application with only the modules of the Java Platform APIs that your application is actuallly using. This will result in smaller application distributables.

### Encapsulation of Internal Packages

A Java module must explicitly tell which Java packages inside the module are to be exported (visible) to other Java modules using the module. A Java module can contain Java packages which are not exported. Classes in unexported packages cannot be used by other Java modules. Such packages can only be used internally in the Java module that contains them.

Packages that are not exported are also referred to as hidden packages, or encapsulated packages.

### Startup Detection of Missing Modules

From Java 9 and forward, Java applications must be packaged as Java modules too. Therefore an application module specifies what other modules (Java API modules or third party modules) it uses. Therefore the Java VM can check the whole module dependency graph from the application module and forward, when the Java VM starts up. If any required modules are not found at startup, the Java VM reports the missing module and shuts down.

Before Java 9 missing classes (e.g. from a missing JAR file) would not be detected until the application actually tried to use the missing class. This would happen sometime at runtime - depending on when the application tried to use the missing class.

Having missing modules reported at application startup time is a big advantage compared to at runtime when trying to use the missing module / JAR / class.

## Java Module Basics

Now you know what a Java module is and what the benefits of Java modules are, let us take a look at the basics of Java modules.

### Modules Contain One or More Packages

A Java module is one or more Java packages that belong together. A module could be either a full Java application, a Java Platform API, or a third party API.

### Module Naming

A Java module must be given a unique name. For instance, a valid module name could be

com.jenkov.mymodule

A Java module name follows the same naming rules as Java packages. However, you should not use underscores (\_) in module names (or package names, class names, method names, variable names etc.) from Java 9 and forward, because Java wants to use underscore as a reserved identifier in the future.

It is recommended to name a Java module the same as the name of the root Java package contained in the module - if that is possible (some modules might contain multiple root packages).

### Module Root Directory

Before Java 9 all Java classes for an application or API were nested directly inside a root class directory (which was added to the classpath), or directly inside a JAR file. For instance, the directory structure for the compiled packages of com.jenkov.mymodule would look like this:

com/jenkov/mymodule

A little more graphically, it would look like this:

* com
  + jenkov
    - mymodule

From Java 9 a module must be nested under a root directory with the same name as the module. In the example above we have a directory structure for a package named com.jenkov.mymodule . This Java package is to be contained within a Java module with the same name ( also com.jenkov.mymodule ).

The directory structure for the above Java package contained in a Java module of the same name, would look like this:

com.jenkov.mymodule/com/jenkov/mymodule

A little more graphically, it would look like this:

* com.jenkov.mymodule
  + com
    - jenkov
      * mymodule

Notice the fullstops (.) in the module root directory name. These fullstops need to be there because they are part of the module name! They are not to be interpreted as subdirectory path dividers!

The module root directory is used both for the source files and compiled classes of a Java module. That means, that if your Java project has a source root directory named src/main/java - then each module inside your project will have its own module root directory under src/main/java. For instance:

src/main/java/com.jenkov.module1

src/main/java/com.jenkov.module2

The same directory structure would be seen in the Java compiler's output directory.

It is common to only have one Java module per project. You will still need the module root directory in that case, but the source and compiler output root directories will only contain a single module root directory.

### Module Descriptor (module-info.java)

Each Java module needs a Java module descriptor named module-info.java which has to be located in the corresponding module root directory. For the module root directory src/main/java/com.jenkov.mymodule the path to the module's module descriptor will be src/main/java/com.jenkov.mymodule/module-info.java .

The module descriptor specifies which packages a module exports, and what other modules the module requires. These details will be explained in the following sections. Here is how a basic, empty Java module descriptor looks:

module com.jenkov.mymodule {

}

First is the module keyword, followed by the name of the module, and then a set of curly brackets. The exported packages and required modules will be specified inside the curly brackets.

Notice also how the module descriptor is suffixed .java and yet it uses a hyphen in the file name (module-info.java). Hyphens are not normally allowed in Java class names, but in module descriptor file names they are required!

### Module Exports

A Java module must explicitly export all packages in the module that are to be accessible for other modules using the module. The exported packages are declared in the module descriptor. Here is how a simple export declaration looks inside a module descriptor:

module com.jenkov.mymodule {

exports com.jenkov.mymodule;

}

This example exports the package called com.jenkov.mymodule .

Please note, that only the listed package itself is exported. No "subpackages" of the exported package are exported. That means, that if the mymodule package contained a subpackage named util then the com.jenkov.mymodule.util package is \*not\* exported just because com.jenkov.mymodule is.

To export a subpackage also, you must declare it explicitly in the module descriptor, like this:

module com.jenkov.mymodule {

exports com.jenkov.mymodule;

exports com.jenkov.mymodule.util;

}

You do not have to export the parent package in order to export a subpackage. The following module descriptor exports statement is perfectly valid:

module com.jenkov.mymodule {

exports com.jenkov.mymodule.util;

}

This example only exports the com.jenkov.mymodule.util package, and not the com.jenkov.mymodule package.

### Module Requires

If a Java module requires another module to do its work, that other module must be specified in the module descriptor too. Here is an example of a Java module requires declaration:

module com.jenkov.mymodule {

requires javafx.graphics;

}

This example module descriptor declares that it requires the standard Java module named javafx.graphics.

JAVA I/O FILES

**Java I/O** (Input and Output) is used to process the input and produce the output.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

## Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

**1) System.out:** standard output stream

**2) System.in:** standard input stream

**3) System.err:** standard error stream

## OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

### OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

### InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

## OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

## InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

# Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a [file](https://www.javatpoint.com/java-file-class).

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use [FileWriter](https://www.javatpoint.com/java-filterwriter-class) than FileOutputStream.

|  |  |
| --- | --- |
| **Method** | **Description** |
| protected void finalize() | It is used to clean up the connection with the file output stream. |
| void write(byte[] ary) | It is used to write **ary.length** bytes from the byte [array](https://www.javatpoint.com/array-in-java) to the file output stream. |
| void write(byte[] ary, int off, int len) | It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream. |
| void write(int b) | It is used to write the specified byte to the file output stream. |
| FileChannel getChannel() | It is used to return the file channel object associated with the file output stream. |
| FileDescriptor getFD() | It is used to return the file descriptor associated with the stream. |
| void close() | It is used to closes the file output stream. |

# Java FileInputStream Class

Java FileInputStream class obtains input bytes from a [file](https://www.javatpoint.com/java-file-class). It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use [FileReader](https://www.javatpoint.com/java-filereader-class) class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| int available() | It is used to return the estimated number of bytes that can be read from the input stream. |
| int read() | It is used to read the byte of data from the input stream. |
| int read(byte[] b) | It is used to read up to **b.length** bytes of data from the input stream. |
| int read(byte[] b, int off, int len) | It is used to read up to **len** bytes of data from the input stream. |
| long skip(long x) | It is used to skip over and discards x bytes of data from the input stream. |
| FileChannel getChannel() | It is used to return the unique FileChannel object associated with the file input stream. |
| FileDescriptor getFD() | It is used to return the [FileDescriptor](https://www.javatpoint.com/java-filedescriptor-class) object. |
| protected void finalize() | It is used to ensure that the close method is call when there is no more reference to the file input stream. |
| void close() | It is used to closes the [stream](https://www.javatpoint.com/java-8-stream). |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| JAVA EXCEPTION HANDLING Exception Handling in Java  1. [Exception Handling](https://www.javatpoint.com/exception-handling-in-java) 2. [Advantage of Exception Handling](https://www.javatpoint.com/exception-handling-in-java#exceptionad) 3. [Hierarchy of Exception classes](https://www.javatpoint.com/exception-handling-in-java#exceptionhierarchy) 4. [Types of Exception](https://www.javatpoint.com/exception-handling-in-java#exceptiontypes) 5. [Exception Example](https://www.javatpoint.com/exception-handling-in-java#exceptionexample) 6. [Scenarios where an exception may occur](https://www.javatpoint.com/exception-handling-in-java#exceptionscenarios)   The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.  In this page, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions. What is Exception in Java **Dictionary Meaning:** Exception is an abnormal condition.  In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. What is Exception Handling Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc. Advantage of Exception Handling The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:   1. statement 1; 2. statement 2; 3. statement 3; 4. statement 4; 5. statement 5;//exception occurs 6. statement 6; 7. statement 7; 8. statement 8; 9. statement 9; 10. statement 10;   Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java. Hierarchy of Java Exception classes The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:  hierarchy of exception handling Types of Java Exceptions There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:   1. Checked Exception 2. Unchecked Exception 3. Error   Types of Java Exceptions Difference between Checked and Unchecked Exceptions1) Checked Exception The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time. 2) Unchecked Exception The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime. 3) Error Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc. Java Exception Keywords There are 5 keywords which are used in handling exceptions in Java.   |  |  | | --- | --- | | **Keyword** | **Description** | | try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. | | catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. | | finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. | | throw | The "throw" keyword is used to throw an exception. | | throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |  Java Exception Handling Example Let's see an example of Java Exception Handling where we using a try-catch statement to handle the exception.   1. public class JavaExceptionExample{ 2. public static void main(String args[]){ 3. try{ 4. //code that may raise exception 5. int data=100/0; 6. }catch(ArithmeticException e){System.out.println(e);} 7. //rest code of the program 8. System.out.println("rest of the code..."); 9. } 10. }   [Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=JavaExceptionExample" \t "_blank)  Output:  Exception in thread main java.lang.ArithmeticException:/ by zero  rest of the code...  In the above example, 100/0 raises an ArithmeticException which is handled by a try-catch block. Common Scenarios of Java Exceptions There are given some scenarios where unchecked exceptions may occur. They are as follows: 1) A scenario where ArithmeticException occurs If we divide any number by zero, there occurs an ArithmeticException.   1. int a=50/0;//ArithmeticException  2) A scenario where NullPointerException occurs If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.   1. String s=null; 2. System.out.println(s.length());//NullPointerException  3) A scenario where NumberFormatException occurs The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.   1. String s="abc"; 2. int i=Integer.parseInt(s);//NumberFormatException  4) A scenario where ArrayIndexOutOfBoundsException occurs If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:   1. int a[]=new int[5]; 2. a[10]=50; //ArrayIndexOutOfBoundsException  Java Exceptions Index  1. [Java Try-Catch Block](https://www.javatpoint.com/try-catch-block) 2. [Java Multiple Catch Block](https://www.javatpoint.com/multiple-catch-block-in-java) 3. [Java Nested Try](https://www.javatpoint.com/nested-try-block) 4. [Java Finally Block](https://www.javatpoint.com/finally-block-in-exception-handling) 5. [Java Throw Keyword](https://www.javatpoint.com/throw-keyword) 6. [Java Exception Propagation](https://www.javatpoint.com/exception-propagation) 7. [Java Throws Keyword](https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws) 8. [Java Throw vs Throws](https://www.javatpoint.com/difference-between-throw-and-throws-in-java) 9. [Java Final vs Finally vs Finalize](https://www.javatpoint.com/difference-between-final-finally-and-finalize) 10. [Java Exception Handling with Method Overriding](https://www.javatpoint.com/exception-handling-with-method-overriding) 11. [Java Custom Exceptions](https://www.javatpoint.com/custom-exception)   Next Topic[Java Try catch block](https://www.javatpoint.com/try-catch-block)  [← prev](https://www.javatpoint.com/java-regex) [next →](https://www.javatpoint.com/try-catch-block) |

### Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

## Difference between Checked and Unchecked Exceptions

### 1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

## Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |

JAVA OBJECT ORIENTED

In this page, we will learn about the basics of OOPs. Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

**Simula** is considered the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.

**Smalltalk** is considered the first truly object-oriented programming language.

The popular object-oriented languages are [Java](https://www.javatpoint.com/java-tutorial), [C#](https://www.javatpoint.com/c-sharp-tutorial), [PHP](https://www.javatpoint.com/php-tutorial), [Python](https://www.javatpoint.com/python-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc.

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

## OOPs (Object-Oriented Programming System)

**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)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

* Coupling
* Cohesion
* Association
* Aggregation
* Composition

## Object



Any entity that has state and behavior 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 behaviors 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 behaviors 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 different ways, 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.

### Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

### Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

### Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

* One to One
* One to Many
* Many to One, and
* Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

### Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a has-a relationship in Java. Like, inheritance represents the is-a relationship. It is another way to reuse objects.

### Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

## Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

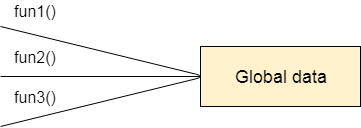


Figure: Data Representation in Procedure-Oriented Programming

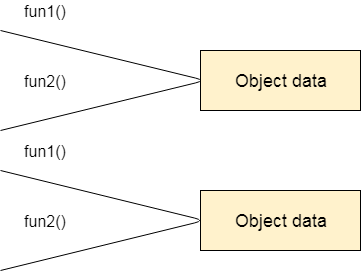


Figure: Data Representation in Object-Oriented Programming

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

## What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

JAVA MYSQL DATABASE ACCESS

# Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

### Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

### 1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

|  |
| --- |
| Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here. |

### 2) Set classpath:

|  |
| --- |
| There are two ways to set the classpath:   * temporary * permanent |

AIRLINE RESERVATION SYSTEM:

Airline reservation system is an integrated passenger processing system. This system includes:

1. Fares
2. Inventory
3. Enquiry
4. Reservations

THIS PROJECT IS CONSTRUCTED BY USING ELLIPSE AND WORKBENCH MYSQL.

OUTPUTS:

