Java

Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

**“extends”** is the keyword used to inherit the properties of a class.

# Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic.

# Exceptions

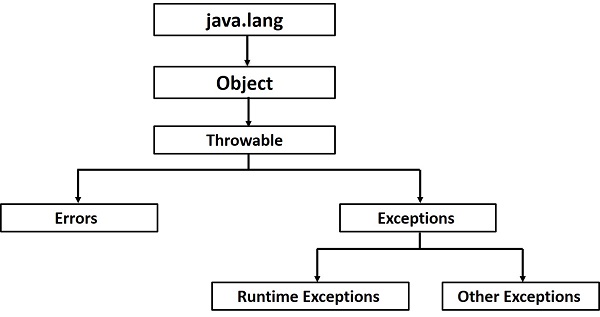
An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

**Checked exceptions** − A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as **compile time exceptions**. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

**Unchecked exceptions** − An unchecked exception is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions**. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

**Errors** − These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.



## **Exceptions Methods**

Following is the list of important methods available in the Throwable class.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **public String getMessage()**  Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor. |
| 2 | **public Throwable getCause()**  Returns the cause of the exception as represented by a Throwable object. |
| 3 | **public String toString()**  Returns the name of the class concatenated with the result of getMessage(). |
| 4 | **public void printStackTrace()**  Prints the result of toString() along with the stack trace to System.err, the error output stream. |
| 5 | **public StackTraceElement [] getStackTrace()**  Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack. |
| 6 | **public Throwable fillInStackTrace()**  Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace. |

**Difference between collections and collection in java**

* Major difference between Collection and Collections is Collection is an interface and Collections is a class.
* Both are belongs to java.util package
* Collection is base interface for list set and queue.
* Collections is a class and it is called utility class.
* Collections utility class contains some predefined methods so that we can use while working with Collection type of classes(treeset, arraylist, linkedlist etc.)
* Collection is base interface for List , Set and Queue.
* **Collection** is a root level interface of the Java Collection Framework. Most of the classes in Java Collection Framework inherit from this interface. **List**, **Set** and **Queue** are main sub interfaces of this interface.
* **Collections** is an utility class in java.util package. It consists of only static methods which are used to operate on objects of type Collection. For example, it has the method to find the maximum element in a collection, it has the method to sort the collection, it has the method to search for a particular element in a collection.
* JDK doesn’t provide any direct implementations of this interface. But, JDK provides direct implementations of it’s sub interfaces. **ArrayList**, **Vector**, **HashSet**, **LinkedHashSet**, **PriorityQueue** are some indirect implementations of Collection interface. **Map interface**, which is also a part of java collection framework, doesn’t inherit from Collection interface. Collection interface is a member of java.util package.

# [What is the difference between an interface and abstract class?](https://stackoverflow.com/questions/1913098/what-is-the-difference-between-an-interface-and-abstract-class)

