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

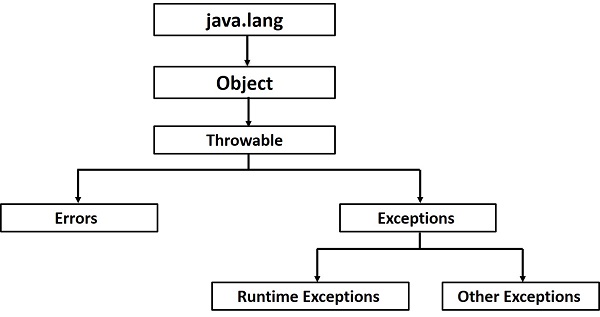
An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

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**

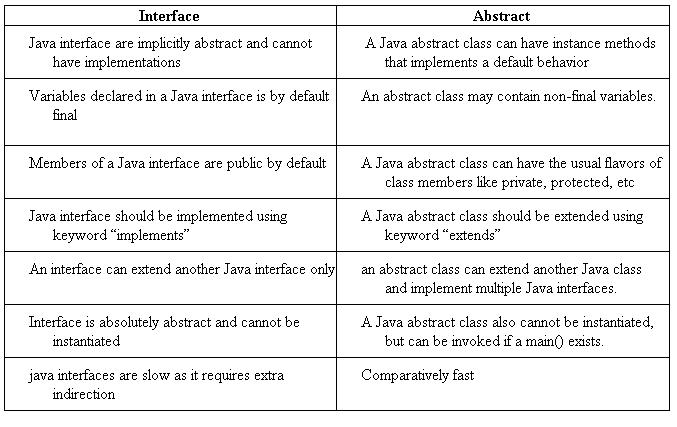
Following is the list of important methods available in the Throwable class.

|  |  |
| --- | --- |
| **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)



## **What is Thread**

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

# How to create thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### **Thread class:**

|  |
| --- |
| Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface. |

### **Commonly used Constructors of Thread class:**

|  |
| --- |
| * Thread() * Thread(String name) * Thread(Runnable r) * Thread(Runnable r,String name) |

### **Commonly used methods of Thread class:**

|  |
| --- |
| 1. **public void run():**is used to perform action for a thread. 2. **public void start():**starts the execution of the thread.JVM calls the run() method on the thread. 3. **public void sleep(long miliseconds):**Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. 4. **public void join():**waits for a thread to die. 5. **public void join(long miliseconds):**waits for a thread to die for the specified miliseconds. 6. **public int getPriority():**returns the priority of the thread. 7. **public int setPriority(int priority):**changes the priority of the thread. 8. **public String getName():**returns the name of the thread. 9. **public void setName(String name):**changes the name of the thread. 10. **public Thread currentThread():**returns the reference of currently executing thread. 11. **public int getId():**returns the id of the thread. 12. **public Thread.State getState():**returns the state of the thread. 13. **public boolean isAlive():**tests if the thread is alive. 14. **public void yield():**causes the currently executing thread object to temporarily pause and allow other threads to execute. 15. **public void suspend():**is used to suspend the thread(depricated). 16. **public void resume():**is used to resume the suspended thread(depricated). 17. **public void stop():**is used to stop the thread(depricated). 18. **public boolean isDaemon():**tests if the thread is a daemon thread. 19. **public void setDaemon(boolean b):**marks the thread as daemon or user thread. 20. **public void interrupt():**interrupts the thread. 21. **public boolean isInterrupted():**tests if the thread has been interrupted. 22. **public static boolean interrupted():**tests if the current thread has been interrupted. |

### **Runnable interface:**

|  |
| --- |
| The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run(). |

|  |
| --- |
| 1. **public void run():**is used to perform action for a thread. |

### **Starting a thread:**

|  |
| --- |
| **start() method** of Thread class is used to start a newly created thread. It performs following tasks:   * A new thread starts(with new callstack). * The thread moves from New state to the Runnable state. * When the thread gets a chance to execute, its target run() method will run. |

### **1) Java Thread Example by extending Thread class**

1. **class** Multi **extends** Thread{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
5. **public** **static** **void** main(String args[]){
6. Multi t1=**new** Multi();
7. t1.start();
8. }
9. }

Output:thread is running...

### **2) Java Thread Example by implementing Runnable interface**

1. **class** Multi3 **implements** Runnable{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
6. **public** **static** **void** main(String args[]){
7. Multi3 m1=**new** Multi3();
8. Thread t1 =**new** Thread(m1);
9. t1.start();
10. }
11. }

Output:thread is running...

## Fundamental Steps in JDBC

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

* Import JDBC packages.
* Load and register the JDBC driver.
* Open a connection to the database.
* Create a statement object to perform a query.
* Execute the statement object and return a query resultset.
* Process the resultset.
* Close the resultset and statement objects.
* Close the connection.

import oracle.jdbc.driver.\*;

import oracle.sql.\*;

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn = DriverManager.getConnection(URL, username, passwd);

Connection conn = DriverManager.getConnection(URL);

OR

Connection conn = DriverManager.getConnection

("jdbc:oracle:thin:@training:1521:Oracle",

"oratest", "oratest");

Statement sql\_stmt = conn.createStatement();

ResultSet rset = sql\_stmt.executeQuery

("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");

rset.close();

sql\_stmt.close();

conn.close();

**DDL**

DDL is short name of **Data Definition Language,** which deals with database schemas and descriptions, of how the data should reside in the database.

* [CREATE](http://www.w3schools.in/mysql/php-mysql-create/) - to create a database and its objects like (table, index, views, store procedure, function, and triggers)
* ALTER - alters the structure of the existing database
* DROP - delete objects from the database
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
* COMMENT - add comments to the data dictionary
* RENAME - rename an object

## **DML**

DML is short name of **Data Manipulation Language** which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

* [SELECT](http://www.w3schools.in/mysql/php-mysql-select/) - retrieve data from a database
* [INSERT](http://www.w3schools.in/mysql/php-mysql-insert/) - insert data into a table
* [UPDATE](http://www.w3schools.in/mysql/php-mysql-update/) - updates existing data within a table
* [DELETE](http://www.w3schools.in/mysql/php-mysql-delete/) - Delete all records from a database table
* MERGE - UPSERT operation (insert or update)
* CALL - call a PL/SQL or Java subprogram
* EXPLAIN PLAN - interpretation of the data access path
* LOCK TABLE - concurrency Control

## **DCL**

DCL is short name of **Data Control Language** which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

* GRANT - allow users access privileges to the database
* REVOKE - withdraw users access privileges given by using the GRANT command

## **TCL**

TCL is short name of Transaction Control Language which deals with a transaction within a database.

* COMMIT - commits a Transaction
* ROLLBACK - rollback a transaction in case of any error occurs
* SAVEPOINT - to rollback the transaction making points within groups
* SET TRANSACTION - specify characteristics of the transaction

# SQL DROP Statement:

The SQL DROP command is used to remove an object from the database. If you drop a table, all the rows in the table is deleted and the table structure is removed from the database. Once a table is dropped we cannot get it back, so be careful while using DROP command. When a table is dropped all the references to the table will not be valid.

# SQL TRUNCATE Statement

The SQL TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

# SQL Delete Statement

The DELETE Statement is used to delete rows from a table.

# SQL CREATE TABLE Statement

The CREATE TABLE Statement is used to create tables to store data. Integrity Constraints like primary key, unique key, foreign key can be defined for the columns while creating the table. The integrity constraints can be defined at column level or table level. The implementation and the syntax of the CREATE Statements differs for different RDBMS.

### The Syntax for the CREATE TABLE Statement is:

CREATE TABLE table\_name

(column\_name1 datatype,

column\_name2 datatype,

... column\_nameN datatype);

* ***table\_name*** - is the name of the table.
* ***column\_name1, column\_name2....*** - is the name of the columns
* ***datatype*** - is the datatype for the column like char, date, number etc.

# SQL Joins

SQL Joins are used to relate information in different tables. A Join condition is a part of the sql query that retrieves rows from two or more tables. A SQL Join condition is used in the SQL [*WHERE Clause*](http://beginner-sql-tutorial.com/sql-where-clause.htm) of select, update, delete statements.

## Joins in SQL

The **SQL Syntax** for joining two tables is:

SELECT col1, col2, col3...  
FROM table\_name1, table\_name2   
WHERE table\_name1.col2 = table\_name2.col1;

If a sql join condition is omitted or if it is invalid the join operation will result in a Cartesian product. The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined. For example, if the first table has 20 rows and the second table has 10 rows, the result will be 20 \* 10, or 200 rows. This query takes a long time to execute.

## SQL Joins Example

Lets use the below two tables to explain the sql join conditions.

**Database table "product";**

|  |  |  |  |
| --- | --- | --- | --- |
| **product\_id** | **product\_name** | **supplier\_name** | **unit\_price** |
| 100 | Camera | Nikon | 300 |
| 101 | Television | Onida | 100 |
| 102 | Refrigerator | Vediocon | 150 |
| 103 | Ipod | Apple | 75 |
| 104 | Mobile | Nokia | 50 |

**Database table "order\_items";**

|  |  |  |  |
| --- | --- | --- | --- |
| **order\_id** | **product\_id** | **total\_units** | **customer** |
| 5100 | 104 | 30 | Infosys |
| 5101 | 102 | 5 | Satyam |
| 5102 | 103 | 25 | Wipro |
| 5103 | 101 | 10 | TCS |

SQL Joins can be classified into Equi join and Non Equi join.

**1) SQL Equi joins**

It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.

**For example:** You can get the information about a customer who purchased a product and the quantity of product.

**2) SQL Non equi joins**

It is a sql join condition which makes use of some comparison operator other than the equal sign like >, <, >=, <=

## 1) SQL Equi Joins:

An equi-join is further classified into two categories:   
a) SQL Inner Join   
b) SQL Outer Join

## a) SQL Inner Join:

All the rows returned by the sql query satisfy the sql join condition specified.

### SQL Inner Join Example:

If you want to display the product information for each order the query will be as given below. Since you are retrieving the data from two tables, you need to identify the common column between these two tables, which is the product\_id.

The query for this type of sql joins would be like,

SELECT order\_id, product\_name, unit\_price, supplier\_name, total\_units   
FROM product, order\_items   
WHERE order\_items.product\_id = product.product\_id;

The columns must be referenced by the table name in the join condition, because product\_id is a column in both the tables and needs a way to be identified. This avoids ambiguity in using the columns in the SQL SELECT statement.

The number of join conditions is (n-1), if there are more than two tables joined in a query where 'n' is the number of tables involved. The rule must be true to avoid Cartesian product.

We can also use aliases to reference the column name, then the above query would be like,

SELECT o.order\_id, p.product\_name, p.unit\_price, p.supplier\_name, o.total\_units   
FROM product p, order\_items o   
WHERE o.product\_id = p.product\_id;

## b) SQL Outer Join:

This sql join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables. The sql outer join operator in Oracle is ( + ) and is used on one side of the join condition only.

The syntax differs for different RDBMS implementation. Few of them represent the join conditions as "sql left outer join", "sql right outer join".

If you want to display all the product data along with order items data, with null values displayed for order items if a product has no order item, the sql query for outer join would be as shown below:

SELECT p.product\_id, p.product\_name, o.order\_id, o.total\_units   
FROM order\_items o, product p   
WHERE o.product\_id (+) = p.product\_id;

The output would be like,

|  |  |  |  |
| --- | --- | --- | --- |
| **product\_id** | **product\_name** | **order\_id** | **total\_units** |
| ------------- | ------------- | ------------- | ------------- |
| 100 | Camera |  |  |
| 101 | Television | 5103 | 10 |
| 102 | Refrigerator | 5101 | 5 |
| 103 | Ipod | 5102 | 25 |
| 104 | Mobile | 5100 | 30 |

**NOTE:** If the (+) operator is used in the left side of the join condition it is equivalent to left outer join. If used on the right side of the join condition it is equivalent to right outer join.

## SQL Self Join:

A Self Join is a type of sql join which is used to join a table to itself, particularly when the table has a FOREIGN KEY that references its own PRIMARY KEY. It is necessary to ensure that the join statement defines an alias for both copies of the table to avoid column ambiguity.

The below query is an example of a self join,

SELECT a.sales\_person\_id, a.name, a.manager\_id, b.sales\_person\_id, b.name   
FROM sales\_person a, sales\_person b   
WHERE a.manager\_id = b.sales\_person\_id;

## 2) SQL Non Equi Join:

A Non Equi Join is a SQL Join whose condition is established using all comparison operators except the equal (=) operator. Like >=, <=, <, >

### SQL Non Equi Join Example:

If you want to find the names of students who are not studying either Economics, the sql query would be like, (lets use student\_details table defined earlier.)

SELECT first\_name, last\_name, subject   
FROM student\_details   
WHERE subject != 'Economics'

The output would be something like,

|  |  |  |
| --- | --- | --- |
| **first\_name** | **last\_name** | **subject** |
| ------------- | ------------- | ------------- |
| Anajali | Bhagwat | Maths |
| Shekar | Gowda | Maths |
| Rahul | Sharma | Science |
| Stephen | Fleming | Science |