HELLO !

Here are the Download Links and Official Documentation

Java (Needs to be installed in your system before next session)

1. Download - https://www.oracle.com/in/java/technologies/javase-downloads.html

2. Resources - https://www.baeldung.com/ ,
https://www.youtube.com/channel/UCYt1sfh5464XaDBH0oH_o7Q

IntelliJ (Needs to be installed in your system before next session)

1. Download - https://www.jetbrains.com/idea/download/
How to download on windows : https://www.youtube.com/watch?v=EMLTOMdIz4w
How to download on Mac : https://www.youtube.com/watch?v=8BrZ7CMrNe8

Getting comfortable with using IntelliJ for beginners/eclipse users :
https://www.youtube.com/watch?v=H_XxH66lm3U

Classes and Objects in Java

Classes and Objects are basic concepts of Object Oriented Programming which revolve around real life entities.

Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

Object

It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: Maruti is an object of class Car .

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general, we **can't** create objects of an abstract class or an interface.

```
Dog tuffy;
```

If we declare reference variable(tuffy) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

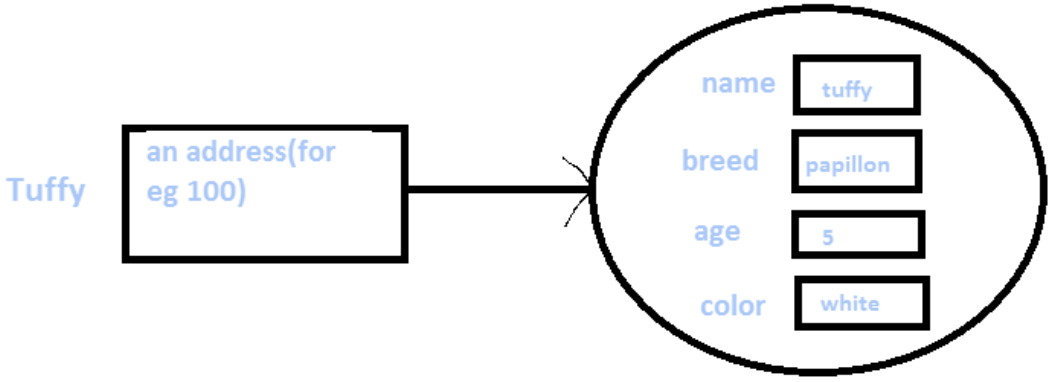
Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

Note : All classes have at least **one** constructor. If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor. This default constructor calls the class parent's no-argument constructor (as it contain only one statement i.e super();), or the *Object* class constructor if the class has no other parent (as Object class is parent of all classes either directly or indirectly).

Lets move to the code now : Dog.java

Create a public class Dog with Instance Variables as String name , String breed ,int age ,String color .
Create a parameterized constructor setting the values of Instance Variables .
Create getters for the instance variables .
Override the toString method to print the instance variables for object of Dog class .
Create a main function in the Dog class and create an object of Dog class using param constructor to set the values .Use that object , call the toString method .



Inheritance in Java

Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.

Java IS-A type of Relationship.

IS-A is a way of saying: This object is a type of that object.

Super Class: The class whose features are inherited is known as superclass(or a base class or a parent class).

Sub Class: The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of

the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

How to use inheritance in Java

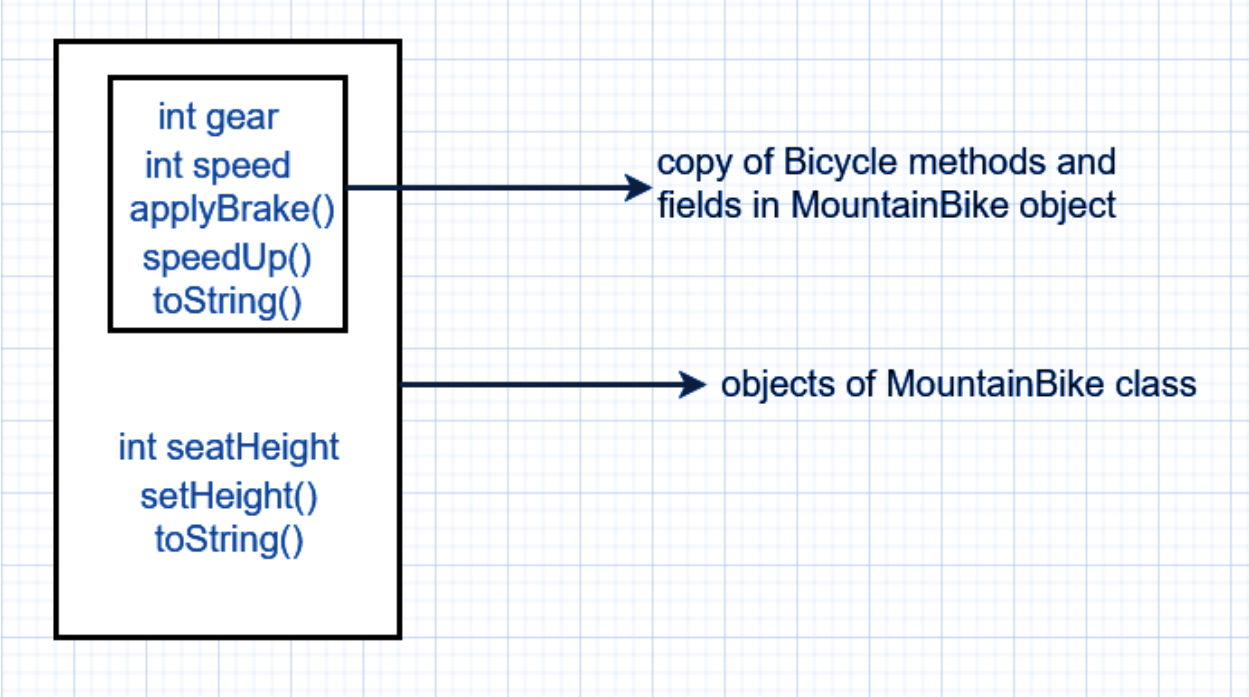
The keyword used for inheritance is **extends**.

Syntax :

```
class derived-class extends base-class
{
   //methods and fields
}
```

Please note that during inheritance only the object of the subclass is created, not the superclass.

Lets dive into code to understand this better : Inheritence.java



Types of Inheritance in Java

Below are the different types of inheritance which are supported by Java.

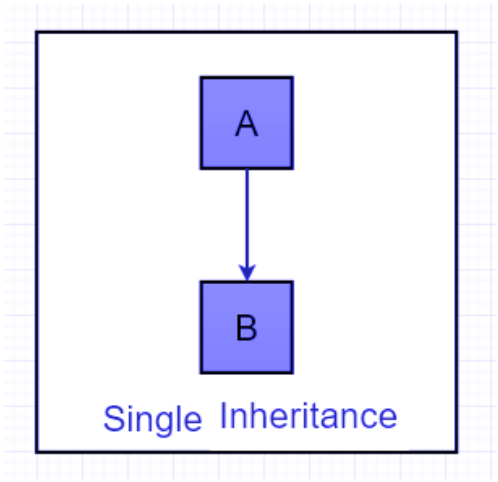
1. Single Inheritance: In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class B.

Lets code :

Create A Vehicle class with function turnHealightOn ,noOfWheels .

Create a child class Car with function noOfWheels .

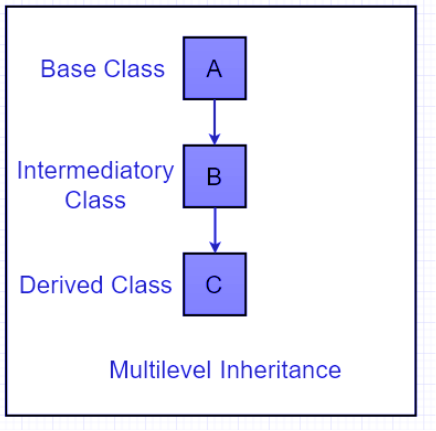
Create a public class Main and create object of Car which can access turn Headlight on .



2. **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.

Lets code :

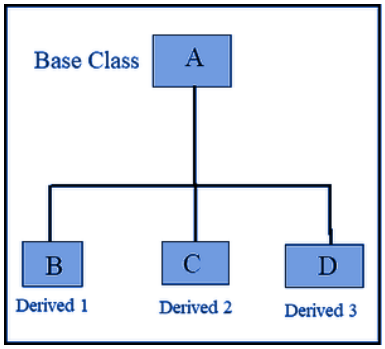
There are 3 classes , one Grandparent , one Parent and one Child . Child can access Parent as well as Grandparent function (MoneyChest1 and MoneyChest2)



3. Hierarchical Inheritance: In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived class B, C and D.

Lets Code

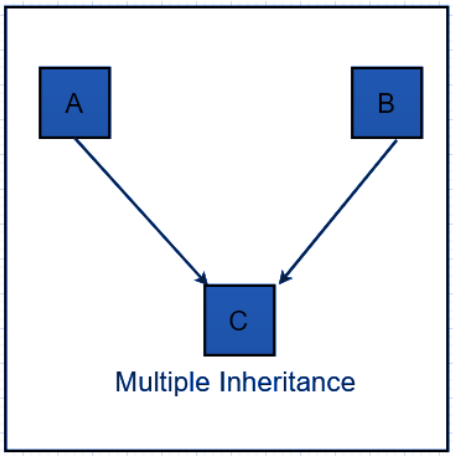
There is one parent A and 3 child classes which extend A . Each child class can access its function and the parent's function but not other children function .



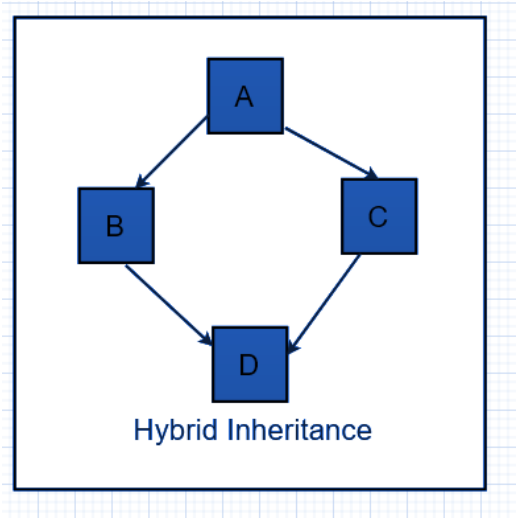
4. **Multiple Inheritance (Through Interfaces):** In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritances with classes. In java, we can achieve multiple inheritances only through Interfaces. In the image below, Class C is derived from interface A and B.

Lets Code :
Create an interface Good with unimplemented function hasBehavour .
Create another interface Better with unimplemented function hasBehavour .

Create a class which implements the above two interfaces and implements the hasBehavour function .
Create a public class and create object of above class .With object of above class , you can call the hasBehavour function .



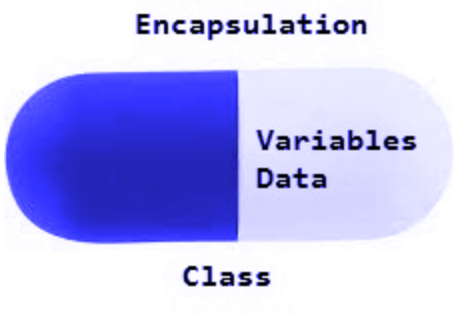
5. Hybrid Inheritance(Through Interfaces): It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritances with classes, hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



Encapsulation in Java

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared.
- As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of a class private, and the class is exposed to the end-user or the world without providing any details behind implementation using the abstraction concept, so it is also known as a **combination of data-hiding and abstraction**.
- Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables



Advantages of Encapsulation:

- **Data Hiding:** The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables. The user will only know that we are passing the values to a setter method and variables are getting initialized with that value.
- **Increased Flexibility:** We can make the variables of the class read-only or write-only depending on our requirement. If we wish to make the variables read-only then we have to omit the setter methods like setName(), setAge(), etc. from the above program or if we wish to make the variables as write-only then we have to omit the get methods like getName(), getAge(), etc. from the above program
- **Reusability:** Encapsulation also improves the re-usability and is easy to change with new requirements.
- **Testing code is easy:** Encapsulated code is easy to test for unit testing.

Lets dive into code now :

Create a class with private instance variables and getters and setters . In another public class , Show setting the value by object of the first class using setters and then print by getters .

Abstraction in Java

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Encapsulation vs Data Abstraction

1. Encapsulation is data hiding(information hiding) while Abstraction is detailed hiding(implementation hiding).
2. While encapsulation groups together data and methods that act upon the data, data abstraction deal with exposing the interface to the user and hiding the details of implementation.

Let's dive into code now :
Abstract Class Shape has a constructor which accepts color as parameter .It has unimplemented abstract functions .Create a child class Rectangle and use the constructor of parent class to set the color property . Remember to not define color in child class . Set it using the constructor by calling the parent constructor

```
abstract class Shape {
	String color;

	// these are abstract methods
	abstract double area();
	public abstract String toString();

	// abstract class can have the constructor
	public Shape(String color)
	{
		System.out.println("Shape constructor called");
		this.color = color;
	}

	// this is a concrete method
	public String getColor() { return color; }
}
```

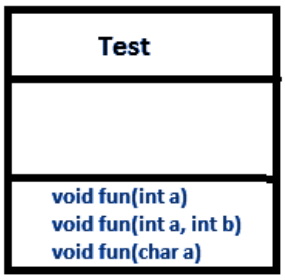
Polymorphism in Java

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

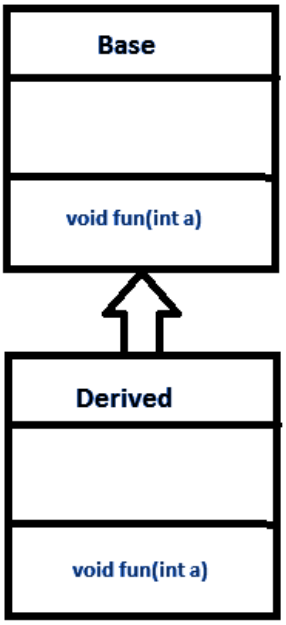
Real life example of polymorphism: A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

In Java polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism



Overloading



Overriding

1. Compile-time polymorphism: It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading. But **Java doesn't support the Operator Overloading**.

Method Overloading: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

Example: By using different types of arguments

Lets code :

Methods with same name but different number of arguments .

2. Runtime polymorphism: It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

Lets code :

```
class Methodoverriding {
        public static void main(String[] args)
        {
                Parent a;
                a = new subclass1();
                a.Print();
                a = new subclass2();
                a.Print();
        }
}
```

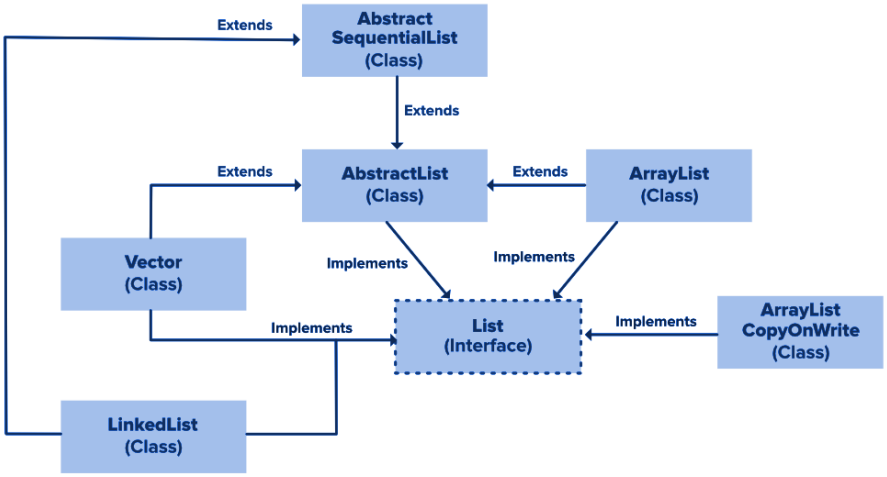
AWESOME !

Here are the Download Links and Official Documentation

Maven **(Needs to be installed in your system before next session)**
1. Download - https://maven.apache.org/install.html
How to download on windows : https://www.youtube.com/watch?v=3ODSQ0EpoQI
How to download on Mac : https://www.youtube.com/watch?v=j0OnSAP-KtU
2. Resources - https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

ArrayList in Java

ArrayList is a part of collection framework and is present in java.util package. It provides us with dynamic arrays in Java. This class is found in java.util package.



Lets Code :
Create a class . Inside the main function , Create an arraylist of type Integer .
Add 10 elements to arraylist using a loop .
Print the arraylist .
Add 100 at position index 2.
Print the arraylist .
Set 200 at position index 2.
Print the arraylist .
Remove the element at index 3
Print the arraylist .
Iterate the ArrayList and print double of each element in the arrayList .
Eg:

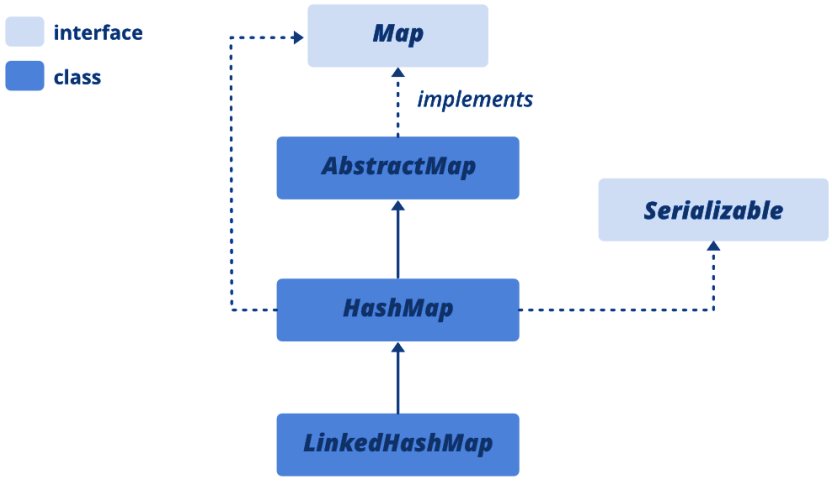
```
for (int num : arrli)
        System.out.print(num + " ");
```

HashMap in Java with Examples

HashMap class is found in java.util package. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs, and you can access them by an index of another type (e.g. an Integer). One object is used as a key (index) to another object (value). If you try to insert the duplicate key, it will replace the element of the corresponding key.

Lets code :
Create a HashMapExample class .
Inside main function, create a hashmap with key as String and object as Integer .
Print the map size .
Now put 4 key value pairs inside Hashmap .
Print the map size .
Print the map .
Remove one of the key-value pair using the map.remove(key) .
Print the Hashmap .
Get the value for one key in the map .
Iterate the hashmap using the for each loop .
Eg:

```
for (Map.Entry<String, Integer> e : map.entrySet())
            System.out.println("Key: " + e.getKey()
                               + " Value: " + e.getValue());
```



EXCEPTION HANDLING

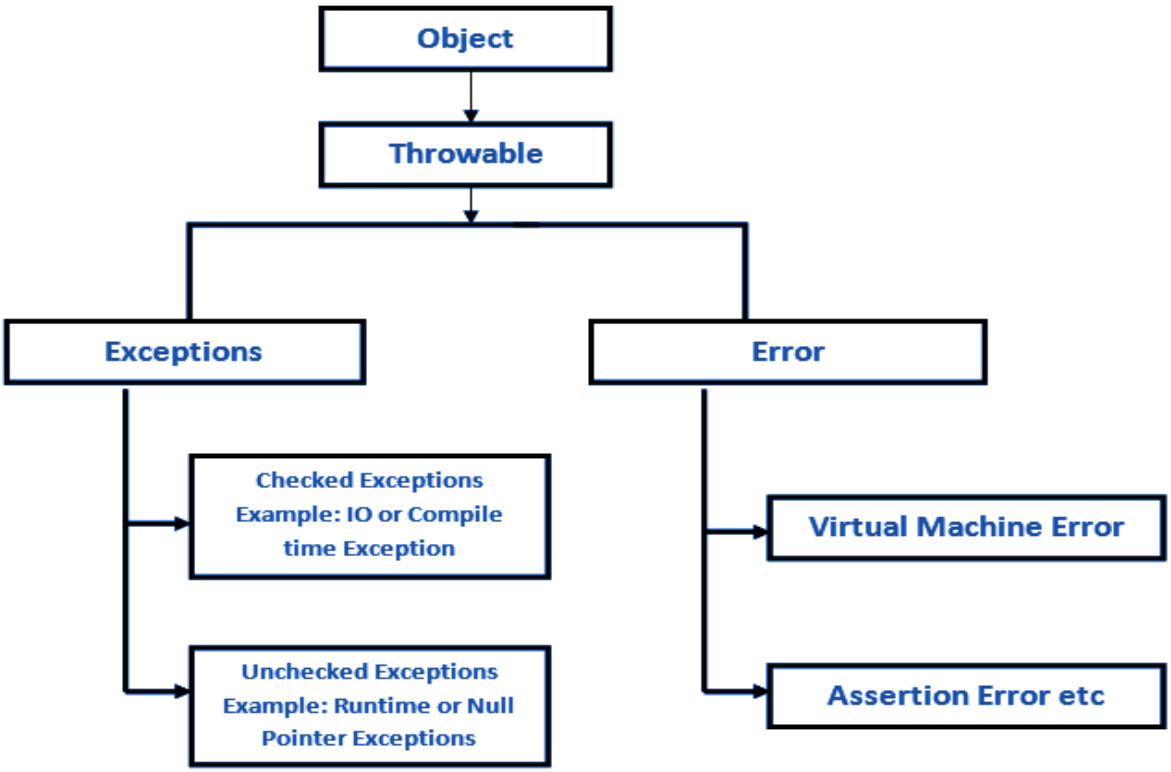
Suppose you are writing a division program in java . You are taking the dividend and divisor from the user as input . If user enters 0 as divisor , then program will give ArithmeticException .
To handle this error , we have exception handling . But how will you handle it ?

```
public static int division(int divd, int div){
   return divd/div;
}
```

WHY EXCEPTION HANDLING ?

1.Separation from Regular code (using try block where business logic is kept with a catch block with handling of possible cases)
2.Propagating the error up in the function call stack
3.Grouping and differentiating Error Conditions

Exception Hierarchy



How to use try-catch clause

```
try {
// block of code to monitor for errors
// the code you think can raise an exception
}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// optional
finally {
// block of code to be executed after try block ends
}
```

Checked vs Unchecked Exceptions in Java

In Java, there are two types of exceptions:

1) **Checked**: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

For example, consider the following Java program that opens file at location "C:\test\a.txt" and prints the first three lines of it. The program doesn't compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

```
import java.io.*;

class Main {
	public static void main(String[] args) {
		FileReader file = new FileReader("C:\\test\\a.txt");
		BufferedReader fileInput = new BufferedReader(file);

		// Print first 3 lines of file "C:\test\a.txt"
		for (int counter = 0; counter < 3; counter++)
			System.out.println(fileInput.readLine());

		fileInput.close();
	}
}
```

Output:

```
Exception in thread "main" java.lang.RuntimeException:
Uncompilable source code -
unreported exception java.io.FileNotFoundException; must be
caught or declared to be
thrown
```

```
    at Main.main(Main.java:5)
```

To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since FileNotFoundException is a subclass of IOException, we can just specify IOException in the throws list and make the above program compiler-error-free.

```
import java.io.*;

class Main {
        public static void main(String[] args) throws IOException {
                FileReader file = new FileReader("C:\\test\\a.txt");
                BufferedReader fileInput = new BufferedReader(file);

                // Print first 3 lines of file "C:\test\a.txt"
                for (int counter = 0; counter < 3; counter++)
                        System.out.println(fileInput.readLine());

                fileInput.close();
        }
}
```

Output: First three lines of file "C:\test\a.txt"

2) **Unchecked** are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

Consider the following Java program. It compiles fine, but it throws *ArithmeticException* when run. The compiler allows it to compile, because *ArithmeticException* is an unchecked exception.

```
class Main {
public static void main(String args[]) {
        int x = 0;
        int y = 10;
        int z = y/x;
}
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: /
by zero
    at Main.main(Main.java:5)
Java Result: 1
```

Exception Handling in java is managed via five keywords: try, catch, throw, throws, and finally.

Keyword	Description
try	This keyword is used to specify a block and this block must be followed by either catch or finally. That is, we can't use try block alone.
catch	This keyword must be preceded by a try block to handle the exception and can be followed by a final block later.
finally	This keyword is used to execute the program, whether an exception is handled or not.
throw	This keyword is used to throw an exception.
throws	This keyword is used to declare exceptions.

throw

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

```
// Java program that demonstrates the use of throw
class Test
{
	public static void main(String[] args)
	{
		System.out.println(1/0);
	}
}
```

throws

In a program, if there is a chance of raising an exception then compiler always warn us about it and compulsorily we should handle that checked exception, Otherwise we will get compile time error saying unreported exception XXX must be caught or declared to be thrown. To prevent this compile time error we can handle the exception in two ways:

1. By using try catch
2. **By using throws keyword**

```
// Java program to demonstrate working of throws
class ThrowsExecp
{
	static void fun() throws IllegalAccessException
	{
```

```
            System.out.println("Inside fun(). ");
            throw new IllegalAccessException("demo");
        }
        public static void main(String args[])
        {
            try
            {
                fun();
            }
            catch(IllegalAccessException e)
            {
                System.out.println("caught in main.");
            }
        }
}
```

Lets recap :

Important points to remember about throws keyword:

- throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
- By the help of throws keyword we can provide information to the caller of the method about the exception

Note : All code files , notes and download links have been uploaded to Github .
Please make sure to download Java in your systems , IntelliJ as IDE and Maven in your systems as build automation tool .