**Abstraction**

Reduce the complexity and increase the efficiency.

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

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

Abstraction lets you focus on what the object does instead of how it does it.

## Why we need an abstract class?

Lets say we have a class Animal that has a method sound() and the subclasses(see [inheritance](https://beginnersbook.com/2013/03/inheritance-in-java/)) of it like Dog, Lion, Horse, Cat etc. Since the animal sound differs from one animal to another, there is no point to implement this method in parent class. This is because every child class must override this method to give its own implementation details, like Lion class will say “Roar” in this method and Dog class will say “Woof”.

each animal must have a sound, by making this method abstract we made it compulsory to the child class to give implementation details to this method. This way we ensures that every animal has a sound.

**Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

# Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

It needs to be extended and its method implemented. It cannot be instantiated.

#### Rule for Abstract class

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have constructors and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

abstract class A{}

### Abstract Method in Java

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

**Example of abstract method**

1. abstract void printStatus();//no method body and abstract

Example

//abstract parent class

abstract class Animal{

//abstract method

public abstract void sound();

}

//Dog class extends Animal class

public class Dog extends Animal{

public void sound(){

System.out.println("Woof");

}

public static void main(String args[]){

Animal obj = new Dog();

obj.sound();

}

}

### Why can’t we create the object of an abstract class?

Because these classes are incomplete, they have abstract methods that have no body so if java allows you to create object of this class then if someone calls the abstract method using that object then What would happen?There would be no actual implementation of the method to invoke.  
Also because an object is concrete. An abstract class is like a template, so you have to extend it and build on it before you can use it.

# Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

## Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



## Simple example of java package

The **package keyword** is used to create a package in java.

## How to access package from another package?

here are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

#### 1) Using packagename.\*

If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

## Example of package that import the packagename.\*

 //save by A.java

 package pack;

 public class A{

   public void msg(){System.out.println("Hello");}

 }

1. //save by B.java
2. package mypack;
3. import pack.\*;
5. class B{
6. public static void main(String args[]){
7. A obj = new A();
8. obj.msg();
9. }
10. }

#### 2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

## Example of package by import package.classname

1. //save by A.java
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
7. //save by B.java
8. package mypack;
9. import pack.A;
11. class B{
12. public static void main(String args[]){
13. A obj = new A();
14. obj.msg();
15. }
16. }

#### 3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

## Example of package by import fully qualified name

1. //save by A.java
2. package pack;
3. public class A{
4. public void msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. package mypack;
8. class B{
9. public static void main(String args[]){
10. pack.A obj = new pack.A();//using fully qualified name
11. obj.msg();
12. }
13. }

# Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

**There are many non-access modifiers,**

such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

**Understanding Java Access Modifiers**

Let's understand the access modifiers in Java by a simple table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

### 1) Private

The private access modifier is accessible only within the class.

### 2) Default

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

### 3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

**Example of protected access modifier**

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

1. //save by A.java
2. package pack;
3. public class A{
4. protected void msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. package mypack;
8. import pack.\*;
10. class B extends A{
11. public static void main(String args[]){
12. B obj = new B();
13. obj.msg();
14. }
15. }

Output:Hello

### 4) Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

**Example of public access modifier**

**Java Access Modifiers with Method Overriding**

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

1. class A{
2. protected void msg(){System.out.println("Hello java");}
3. }
5. public class Simple extends A{
6. void msg(){System.out.println("Hello java");}//C.T.Error
7. public static void main(String args[]){
8. Simple obj=new Simple();
9. obj.msg();
10. }
11. }

Encapsulation

**Encapsulation in Java** is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

Protect the data from unauthorized users.

### Advantage of Encapsulation in Java

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.