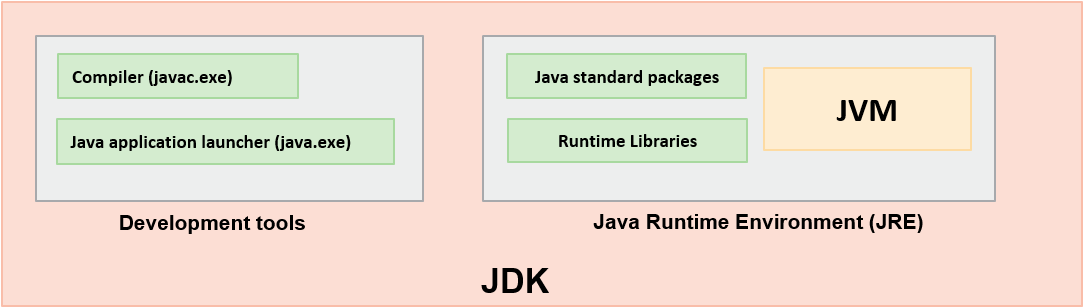
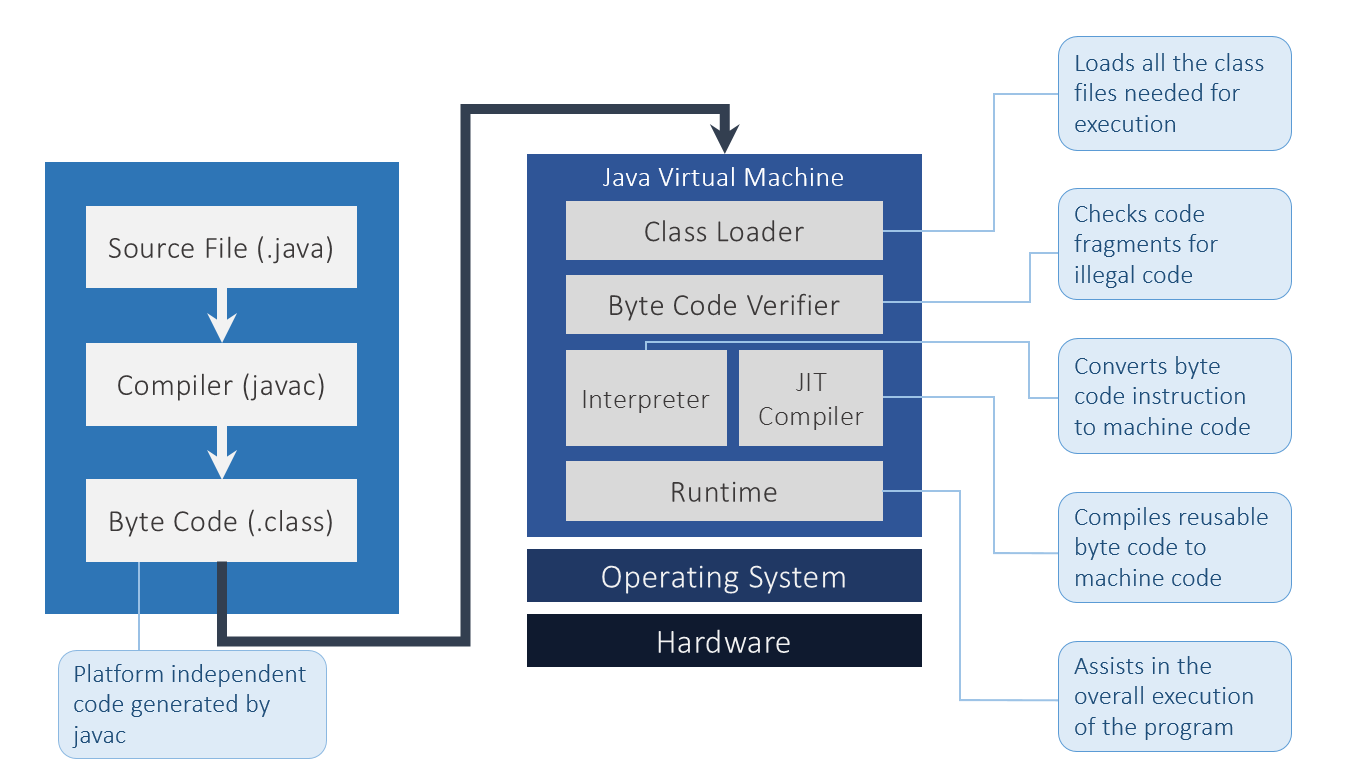
JAVA important notes



This is the basic JAVA development kit scenario and now onwards from java 11 we don’t have JRE

**Best Practices:**

* The name of class should start with an uppercase letter using Camel Case notation, like for example **E**mployee
* The name of methods should start with a lowercase letter using CamelCase notation, like for example **getName()**



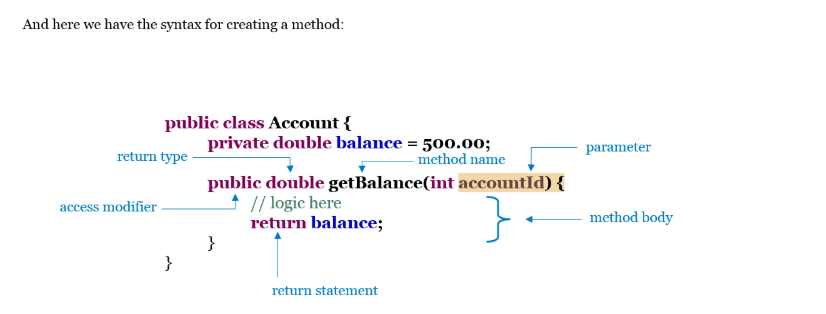
\*\*\*\* **interpreters** are platform dependant

\*\*\*\* **JVM is an Interpreter**

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

OOPS:

An instance of a class contains its own copy of variables (attributes) and the methods (behavior) with all the other objects of that class.

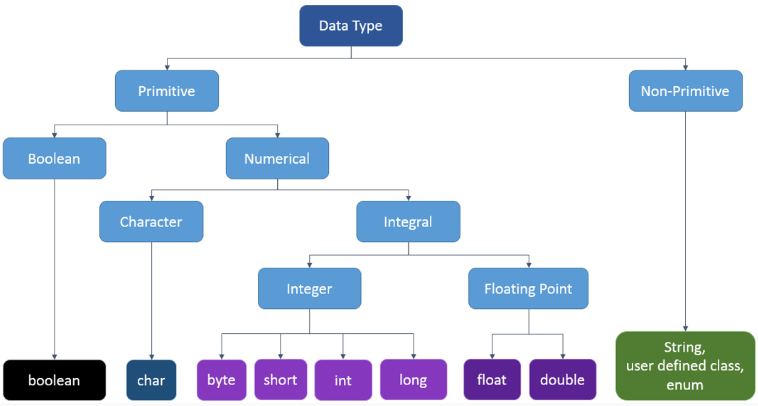
Objects collaborate by exchanging messages with one another via method invocations (i.e., invoking each other's behaviors). 

\*\*\*\* main() method cannot be private as it will result in a RUN TIME error

\*\*\*\* even if there is no main class the .class file will be generated if the class has no errors

Rules for naming identifiers

* Case-sensitive
* Should not start with a number
* Can start with a letter, $ or \_
* Should not have spaces
* Should not be a Java keyword or literal
* No restriction on the length



These are the basic data types of java which helps in defining the variables

Non Primitive Data types holds the memory address where the data item (object) is stored and also known as reference data types

|  |  |  |
| --- | --- | --- |
| **Datatype** | **Default Value** | **Default size** |
| boolean | false\*\*\*\* | 1 bit (depends on JVM) |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

 In Java we can have three different scopes of variables:

**Local Variable**

Variables are declared and used inside same method only.

**Static Variable**

Static variables are declared inside class and can be used outside class with class name and class object. But class name is preferred.

**Instance Variable** Instance variables are declared inside class and can be used outside class with class object only.

Syntax: result = (test\_condition) ? (value\_1) : (value\_2) **{Ternary Operator}**

Now for ++ operator :

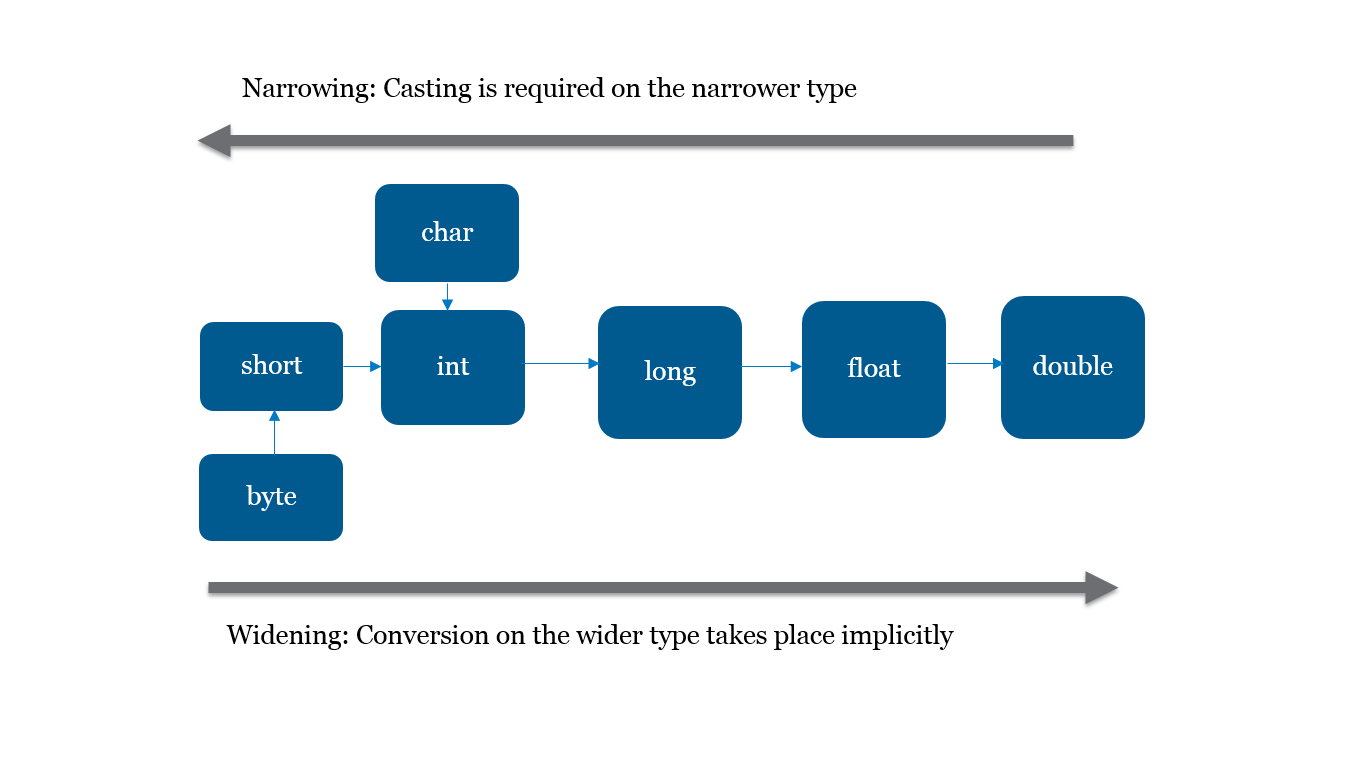
for m++ we first use m then increment its value and for ++m we first increment m then use its value in the expression

Java is a strongly-typed language, where type-checking is strictly enforced at runtime. This makes it impossible to convert incompatible types.

There are 2 types of type conversions:

* Widening conversion / implicit type casting
* Narrowing conversion / explicit type casting

Converting a variable of smaller datatype to the larger datatype, without data loss is known as widening conversion while Converting a variable of larger datatype to the smaller datatype, with some data loss is known as narrowing conversion.



1. double d = 234.04;
2. long l = (long)d; *//explicit type casting*
3. int i = (int)l; *// explicit type casting*
4. int i = 300;
5. long l = i; *//no explicit type casting*
6. float f = l; *//no explicit type casting*

The instanceof operator in Java is used to check whether the object is an instance of the specified class or subclass or interface.

A boolean cannot be cast to any other data type and vice versa

Switch statements can only take Int, String, Enum type for case matching while case can hold expressions in the it like “ Case 1+2: -----”

for(int i: arr) { //for-each loop in java

You can't use for-each when

* You have to modify or delete the collection or array selectively
* You need to iterate through more than one collection in parallel

Classes----------------------------------🡪

If the programmer does not provide the default constructor, the compiler will provide one

On finding a parametrized constructor, compiler doesn’t give the default constructor.

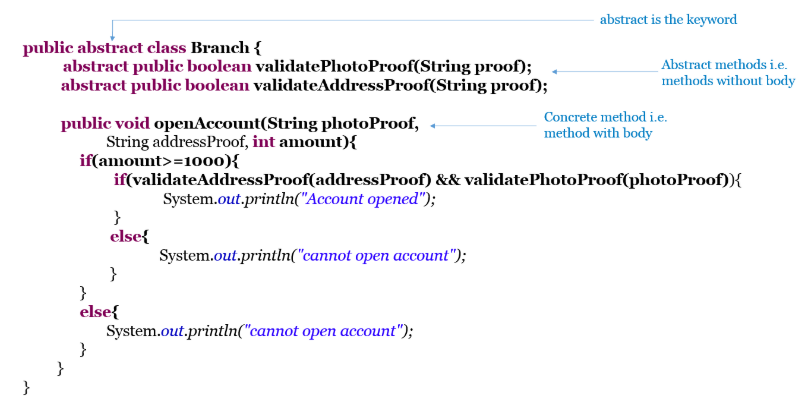
It doesn’t have a return type.

It may or may not have parameters.

We have to always provide a return type in constructor otherwise error will be generated in the code

We also have to see that if suppose void is the return type and something is written in the constructor then also it won’t print in the console as the return type is null

this keyword is used to refer to the current instance of an object.

**ABSTRACTION****--------------------------------------------**

**This is an implementation of abstract class which is basically used for hiding details in a program**

* An abstract class cannot be instantiated using the new keyword.
* The subclass of an abstract class can be only instantiated if it provides the implementation for all the abstract methods.
* If a class has at least one abstract method, then the class must be declared as abstract.
* If the subclass does not implement all the abstract methods, then the subclass must also be declared as abstract.
* An abstract class can also have concrete methods i.e. methods with implementation.
* An abstract class reference can be assigned an object of its subclass, thereby achieving run-time polymorphism.
* Always use abstract classes when you want to share code among several closely related classes
* Always use abstract classes when you want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong
* **An abstract class cannot have private abstract methods and also there is no compulsion that we have to use abstract method in the class , we can also have constructor inside an abstract class**

-------------------------------------------**INTEFACE-----------------------------------------------------------------------------**

**We required interface in java because we do not have multiple inheritance in java so if in case we need to extend new abstract class which is already extending an abstract class we cannot do this here so to ease this possibility we have interfaces it is same but with 100% abstraction**

**public interface IBank { //Interfaces are identified by keyword interface**

**int CAUTION\_MONEY = 2000; //variables are by default public final and static**

**String createAccount(Customer customer);**

**double issueVehicleLoan(String vehicleType, Customer customer);**

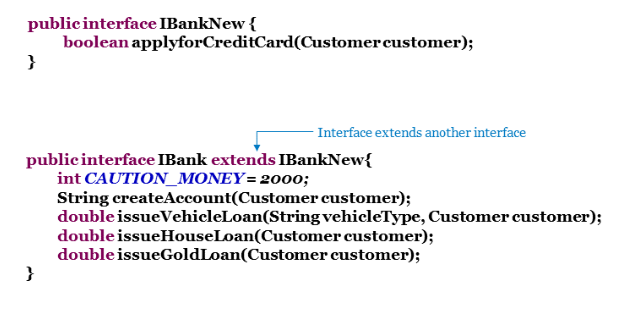
**double issueHouseLoan(customer customer);**

**double issueGoldLoan(Customer customer); //Methods are by default public and abstract**

**}**

* **An interface defines a contract for a class.**
* **Objects can't be created for interface and an interface cannot have private or protected members.**
* **In an interface, all methods are implicitly public and abstract and variables are implicitly public, static and final**
* **The class which implements the interface has to provide definitions for all abstract methods.**
* **If at least one abstract method of the interface has not been overridden by the class that implemented the interface, make it abstract class.**
* **Inheritance is possible in an interface and it supports multiple inheritance.**

public class MumbaiOffice implements IBank, IBankNew{ //This is an example of multiple inheritance using interfaces

\*\*\*\* we can also have inheritance in interfaces using this 

Key point here is that when some class implements the interface in java then it is done by **implements** keyword but we can use **extend** keyword in java interfaces to extend each other to have INHERITANCE in INTERFACES

---------------------------ENCAPSULATION-------------------------------------------------------

**Encapsulation in Java** is a *process of wrapping code and data together into a single unit*

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

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.

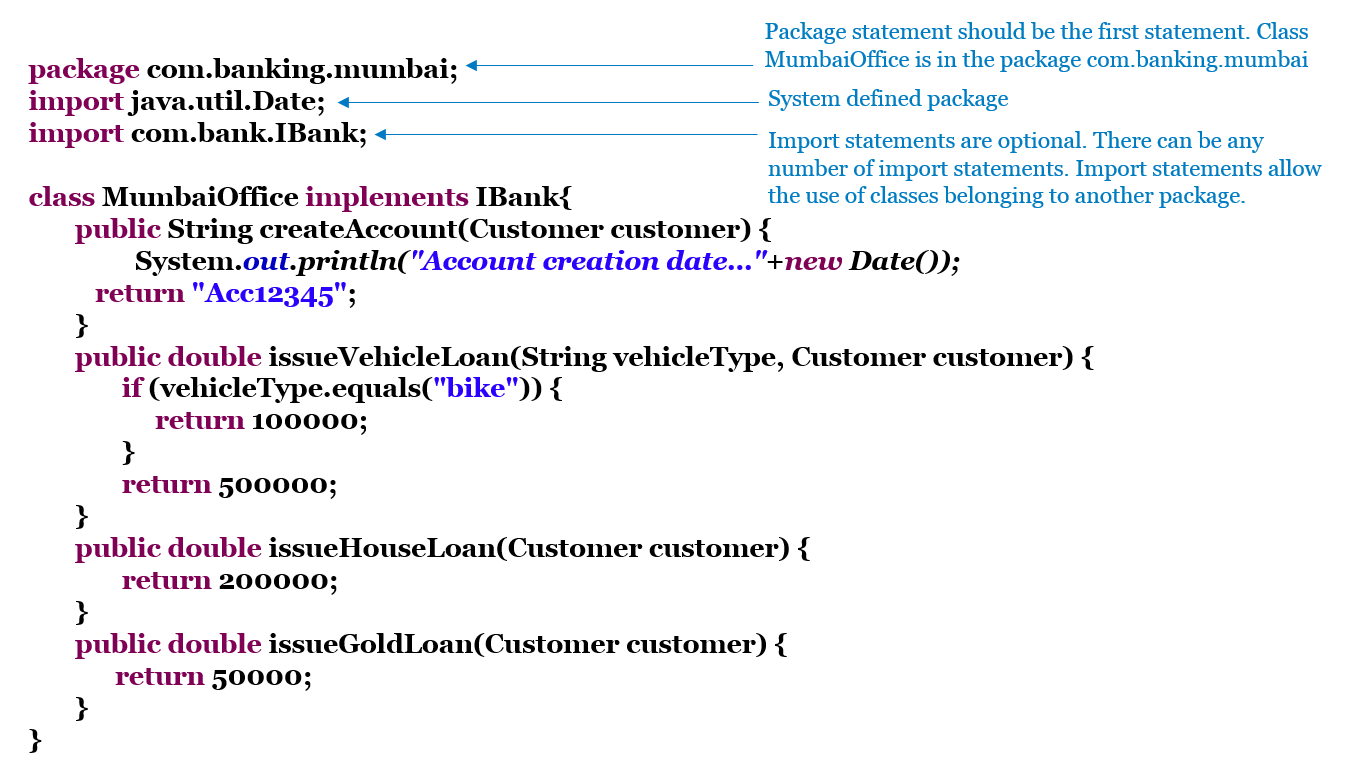
The key point in encapsulation is the use of **SETTERs & Getters and making all the variables private so that data is hidden and cannot be used without the above methods**

------------------------------PACKAGES------------------------------------------------------------

Similarly, a package is a grouping mechanism in Java which contains all related classes and interfaces.  
The package is a folder that contains all these related classes and interfaces.  
They also restrict access to certain classes and interfaces.  
Without packages, we will end up having huge amount of code which is not categorized properly.

Packaging helps in functional grouping of java codes

package com.banking; //this is an example of how we import any package



| **package** | **Description** |
| --- | --- |
| java.lang | Includes classes like String, StringBuffer, Object, System which are fundamental to the Java Programming Language |
| java.util | Includes utility classes like Calendar, Collection, Date |
| java.io | Includes IO classes like FileReader and FileWriter |
| java.net | Includes networking classes like Inet6Address and Socket |
| java.awt | Includes classes to create user interface like Button and Checkbox |

All packages except java.lang package need to be imported explicitly for developing programs.

\*\*\*\*An **import** statement can be used to access the classes and interface of a different package from the current package.

--------------------------------------ACCESS Modifiers-----------------------------------------------------

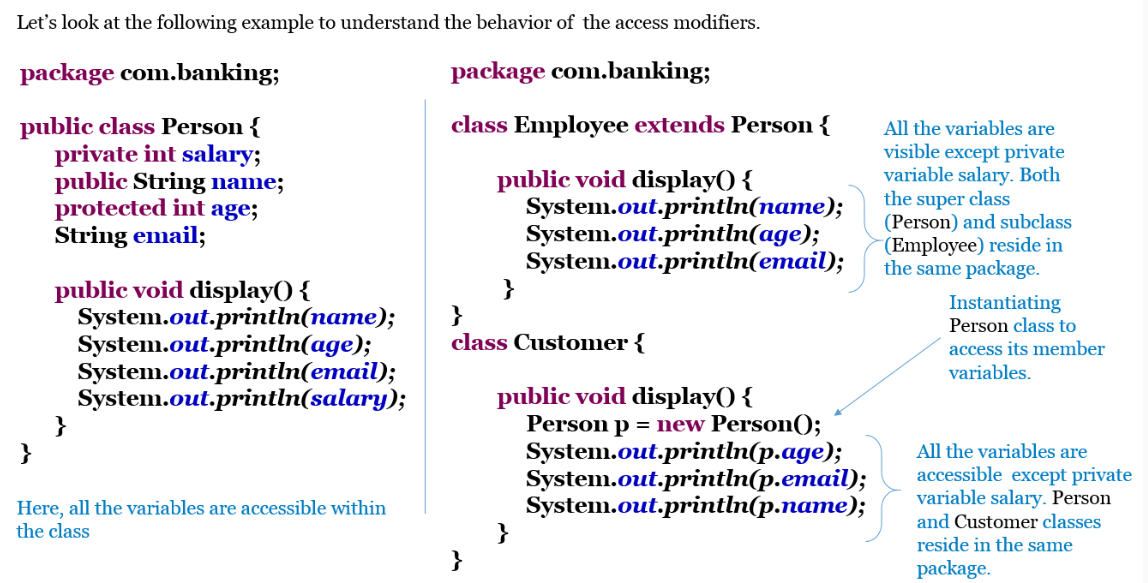
Java provides access protection at 2 levels:  
  
- Class/Interface level  
  
- Member level (variable, method, and constructor)

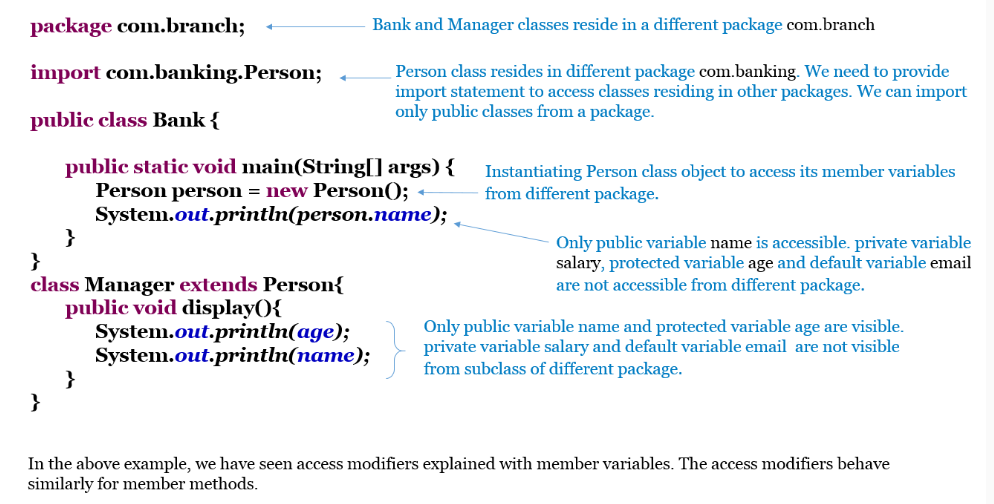
4 types of access modifiers in java are -🡪 private, protected, default and public

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Applicable To** | **Who can Access** |
| private | Data members, methods, and inner class | All members from the same class only |
| (No keyword, usually we call it default) | Data members, methods, classes, and interfaces | All classes from within the same package |
| protected | Data members, methods, and inner class | All classes from within the same package as well as all subclasses i.e. even subclasses residing in a different package |
| public | Data members, methods, classes, and interfaces | Any class(even other packages) |

This is the best table to rectify your query

This is how we access all the queries in java to protect the original data to have restriction as to who can access it (maintains integrity)





INNER Classes->

* A class can host another class called inner class.
* An inner class can contain member methods and member variables just as any other class.
* Members of an instance of the inner class can access an instance of an outer class, because an inner class is just another member of the outer class. Inner classes can also access the outer class's private members.
* An inner class can be static.
* An inner class can have private, protected, public or package access.

class Manager { // Manager class represents outer class

class Grade { // The Grade class represents the inner class

private char grade;

public char calculateGrade(String employeeID, int point) {

//calculate grade

return grade;

}

}

public char checkEmployeeID(String employeeId, int point) {

Grade grade = new Grade(); //Inner class is instantiated and used in the method of the outer class

return grade.calculateGrade(employeeid,point);

}

}

Manager.Grade grade = manager.new Grade(); //this is the way we initialize an inner class like Grade here ouside the outer class scope

ANONYMOUS INNER CLASSES🡪

**interface Grade{**

**char calculateGrade(String employeeID, int point);**

**}**

**class Manager {**

**Grade grade = new Grade(){ //Represents anonymous inner class**

\*\*\*\*It is not mandatory that an anonymous inner class should only implement an interface. It can also be a subclass. But an anonymous inner class cannot implement an interface and extend a class at the same time.

**Object class 🡪**

**Implicit super class inherited by all the classes have methods like** .equals(object) .getClass(), hashCode(), .toString()

These methods can be overridden

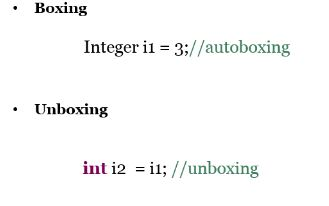
* If object1 and object2 are equal, their hash code values should be same but viceversa is not true

Wrapper Class 🡪

Resides over java.lang.package

| **Primitive Data Type** | **Wrapper Type** |
| --- | --- |
| boolean | Boolean |
| char | Character |
| long | Long |
| float | Float |
| int | Integer |
| short | Short |
| byte | Byte |

* Wrapper class is used to represent primitive data types as Objects.
* Java collections cannot store primitive types. It can store only objects.



Wrapper class has methods that help convert one type of data to another data type like for example **Integer.parseInt()**

1. **int i = 45;*//primitive data int***
2. **Integer integer = new Integer(i);*// Integer wrapper class instantiation***
3. **int i2 = integer.intValue();*// unwrapping primitive data int from wrapper object***

== is used to compare the refrence whereas in .equals we compare the actual value stored in the variable

**SCANNER CLASS🡪**

we can get it inside the java.util.package

| **Methods** | **Description** |
| --- | --- |
| public String next() | Returns the next token from the scanner. |
| public String nextLine() | Moves the scanner position to the next line and returns the value as a string. |
| public byte nextByte() | Scans the next token as a byte. |
| public short nextShort() | Scans the next token as a short value. |
| public int nextInt() | Scans the next token as a int value. |
| public long nextLong() | Scans the next token as a long value. |
| public float nextFloat() | Scans the next token as a float value. |
| public double nextDouble() | Scans the next token as a double value. |

**The garbage collector will free memory occupied by an object under three conditions:**

* **When the reference of the object is set to null.**
* **When the reference of the object is set to some new object such that no reference points to the previous object. In this class, the un-referenced previous object will be garbage collected.**
* **When a reference is local to some method then it will be removed from the Stack as soon as the method is executed. If this reference is pointing to an object, that object will also be garbage collected.**

The finalize method written in a Java class is called garbage collection

* The finalize method is used when certain objects are required to perform some action before they are destroyed.
* By overriding this method, we can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.