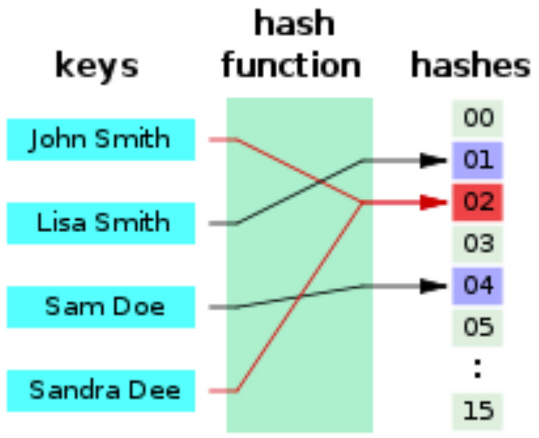
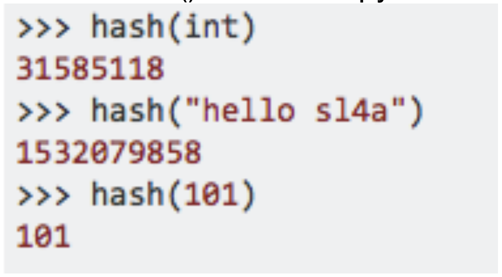
———————— Hash Functions

A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes. One use is a data structure called a hash table, widely used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file. An example is finding similar stretches in DNA sequences. They are also useful in cryptography.



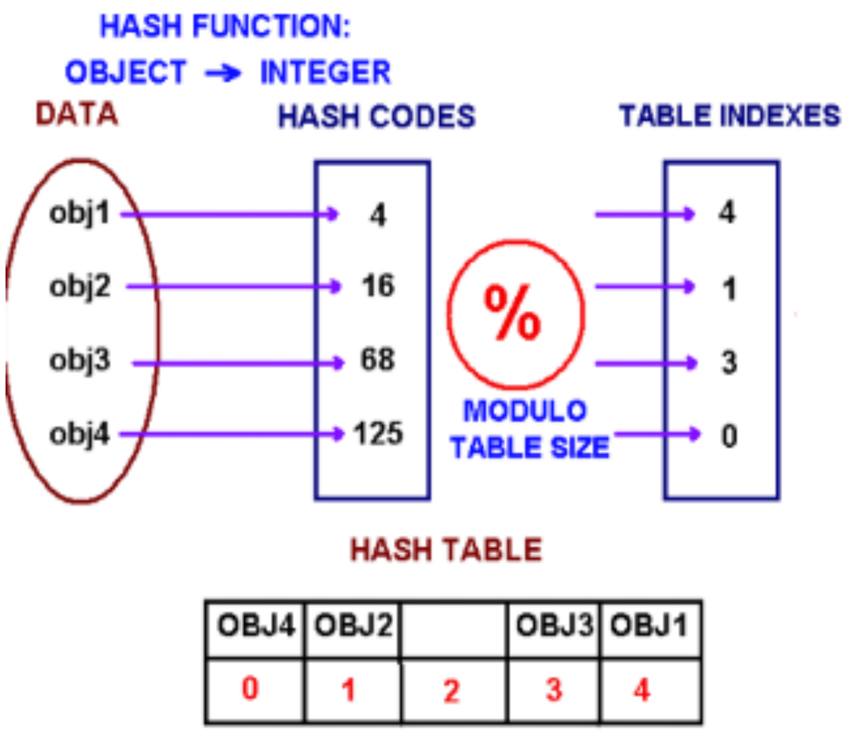
A hash function that maps names to integers from 0 to 15. There is a collision between keys "John Smith" and "Sandra Dee".

Here is hash() function in python



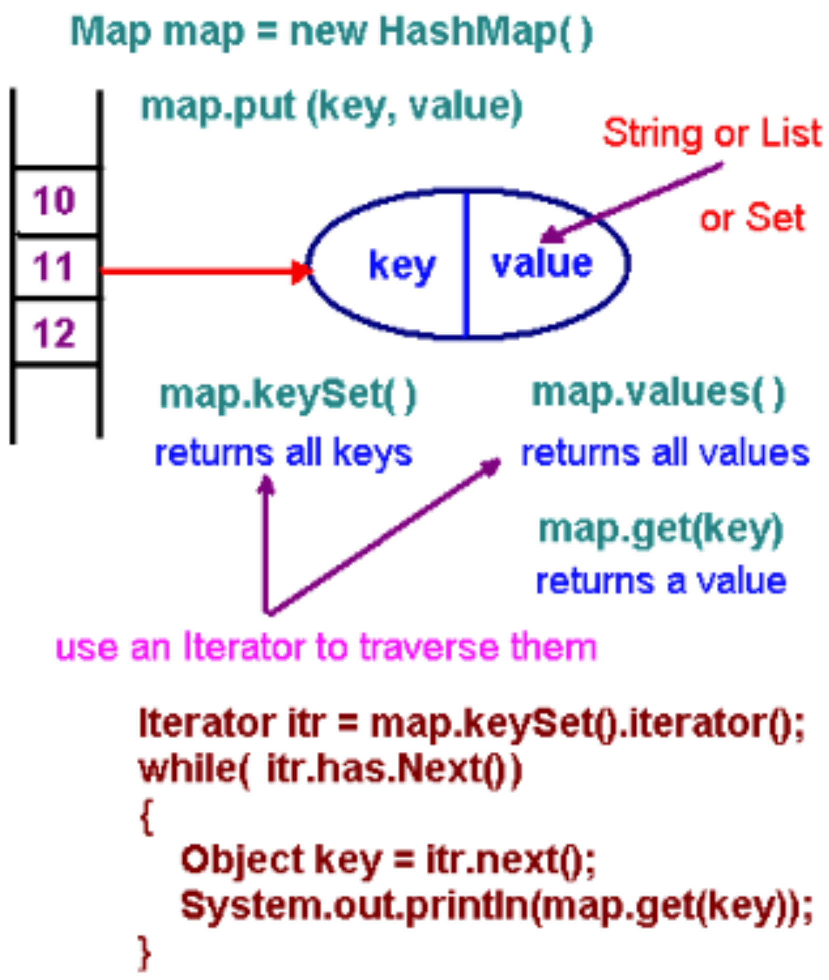
```
>>> hash(int)
31585118
>>> hash("hello sl4a")
1532079858
>>> hash(101)
101
```

The problem at hands is to speed up searching. Consider the problem of searching an array for a given value. If the array is not sorted, the search might require examining each and all elements of the array. If the array is sorted, we can use the binary search, and therefore reduce the worse-case runtime complexity to O(log n). We could search even faster if we know in advance the index at which that value is located in the array. Suppose we do have that magic function that would tell us the index for a given value. With this magic function our search is reduced to just one probe, giving us a constant runtime O(1). Such a function is called a hash function . A hash function is a function which when given a key, generates an address in the table.



```
Integer obj1 = new Integer(2009);
String obj2 = new String("2009");
System.out.println("hashCode for an integer is " + obj1.hashCode());
System.out.println("hashCode for a string is " + obj2.hashCode());
```

HashMap is a collection class that is designed to store elements as key-value pairs. Maps provide a way of looking up one thing based on the value of another.

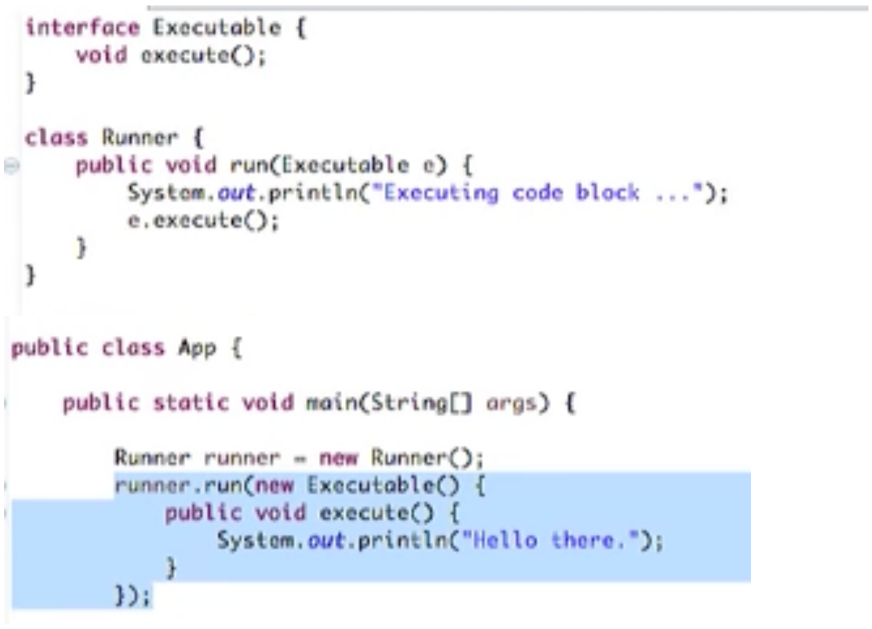


```
Iterator itr = map.keySet().iterator();
while( itr.has.Next())
{
   Object key = itr.next();
   System.out.println(map.get(key));
}
```

```
String[] data = new String("Nothing is as easy as it looks").split(" ");

HashMap<String, Integer> hm = new HashMap<String, Integer>();

for (String key : data)
{
      Integer freq = hm.get(key);
      if(freq == null) freq = 1; else freq ++;
      hm.put(key, freq);
}
System.out.println(hm);
```

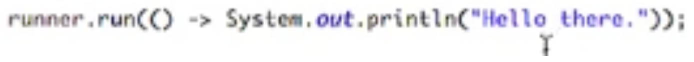
This prints `{as=2, Nothing=1, it=1, easy=1, is=1, looks=1}`.

What is Lambda Express — So in java , passing block of a code to a method is called Lambda expression . So as shown below, method execute is block of code . This block of code is called from class by first creating an object e of an interface Executable. And then to run this , we are calling another class called App and then creating object of Runner and then printing something.. This whole process is lengthy code which can be avoided by use of Lambda expression



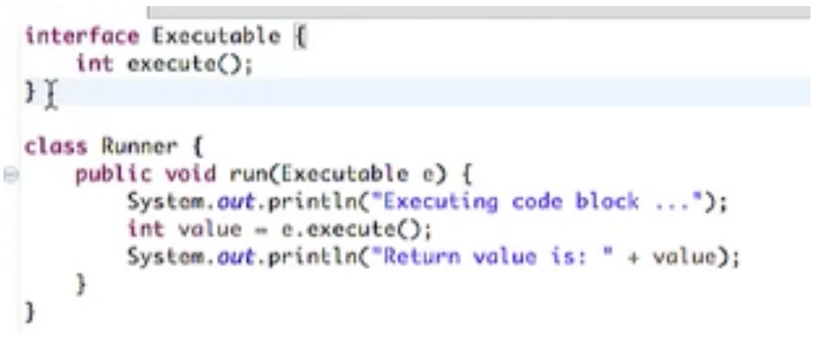
```
interface Executable {
    void execute();
}

class Runner {
    public void run(Executable e) {
        System.out.println("Executing code block ...");
        e.execute();
    }
}

public class App {

    public static void main(String[] args) {

        Runner runner = new Runner();
        runner.run(new Executable() {
            public void execute() {
                System.out.println("Hello there.");
            }
        });
```

So then how to use Lambda expression is shown below ——
Instead of using that highlight block above, use below line . This is syntax of lambda expression

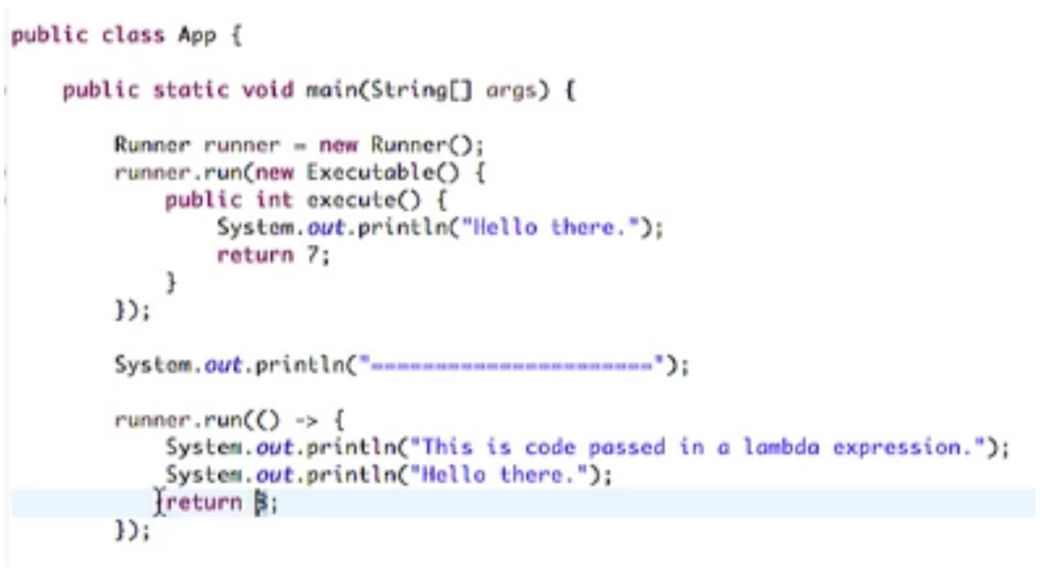


```
runner.run(() -> System.out.println("Hello there."));
```

In example above, return type of Executor method was void .. now in example below, return type of Executor is int. Even then no need to specify that in the lambda expression. It knows that return type automatically …

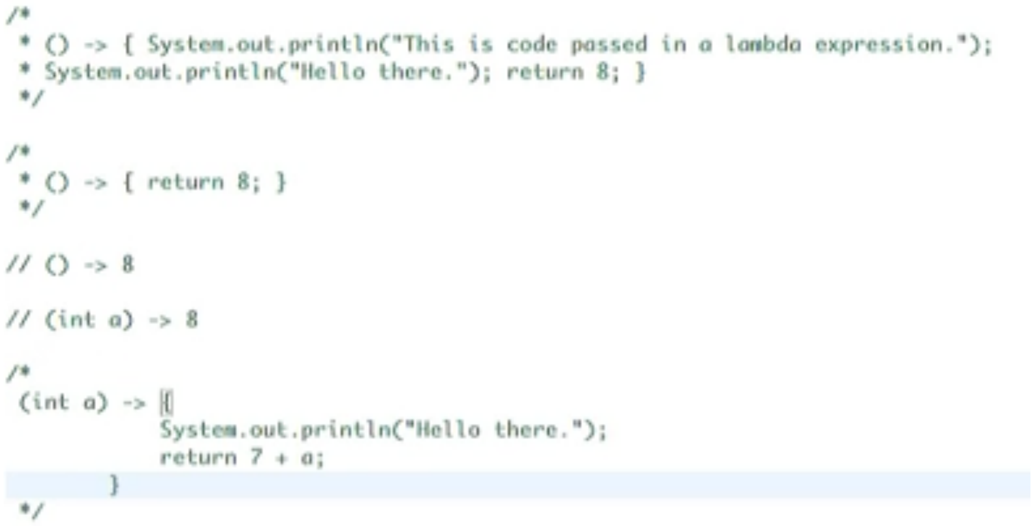


```
interface Executable {
    int execute();
}

class Runner {
    public void run(Executable e) {
        System.out.println("Executing code block ...");
        int value = e.execute();
        System.out.println("Return value is: " + value);
    }
}
```



```java
public class App {

    public static void main(String[] args) {

        Runner runner = new Runner();
        runner.run(new Executable() {
            public int execute() {
                System.out.println("Hello there.");
                return 7;
            }
        });

        System.out.println("=====================");

        runner.run(() -> {
            System.out.println("This is code passed in a lambda expression.");
            System.out.println("Hello there.");
            return 8;
        });
```

— Here is summary of all lambda expressions



```java
/*
 * () -> { System.out.println("This is code passed in a lambda expression.");
 * System.out.println("Hello there."); return 8; }
 */

/*
 * () -> { return 8; }
 */

// () -> 8

// (int a) -> 8

/*
 (int a) -> {
             System.out.println("Hello there.");
             return 7 + a;
        }
 */
```

Here is great video on lambda expression - https://www.youtube.com/watch?v=q5i_O4Uj_O8

—— what is build path - The build path is the classpath that is used for building a Java project. A classpath is simply an array of classpath entries (IClasspathEntry) that describe the types that are available.

——————————

How to create your own RPM package -
https://www.youtube.com/watch?v=n8gwjpYxt2A

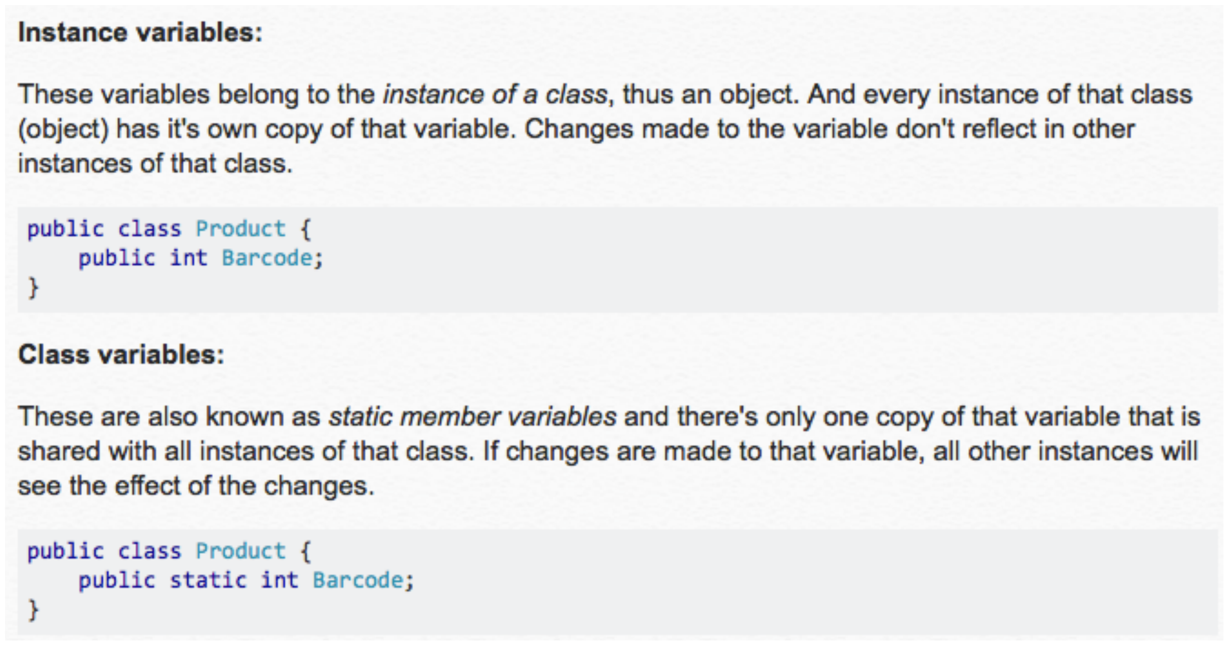
Few linux concepts
http://www.linuxtechi.com/experience-linux-admin-interview-questions/

——————————— Java interview questions
Main concept is class is prototype and when we use new keyword to instantiate it , we actually create an object from that prototype. Constructor always get run when we create fresh object.

Some features include Object Oriented, Platform Independent, Robust, Interpreted, Multi-threaded
Java Architectural Neutral ——— It's compiler generates an architecture-neutral object file format, which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
How Java enable high performance —— Java uses Just-In-Time compiler to enable high performance. Just-In-Time compiler is a program that turns Java bytecode, which is a program that contains instructions that must be interpreted into instructions that can be sent directly to the processor.

What is JAVA virtual machine **(it makes the java code platform agnostic**) ——When Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run. Object is a runtime entity and it's state is stored in fields and behavior is shown via methods. A class is a blue print from which individual objects are created. A class consist of Local variable, instance variables and class variables.

—Variables defined inside methods, constructors or blocks are called local variables.
— Instance variables are variables within a class but outside any method.
— These are variables declared with in a class, outside any method, with the static keyword.



Instance variables:

These variables belong to the *instance of a class*, thus an object. And every instance of that class (object) has it's own copy of that variable. Changes made to the variable don't reflect in other instances of that class.

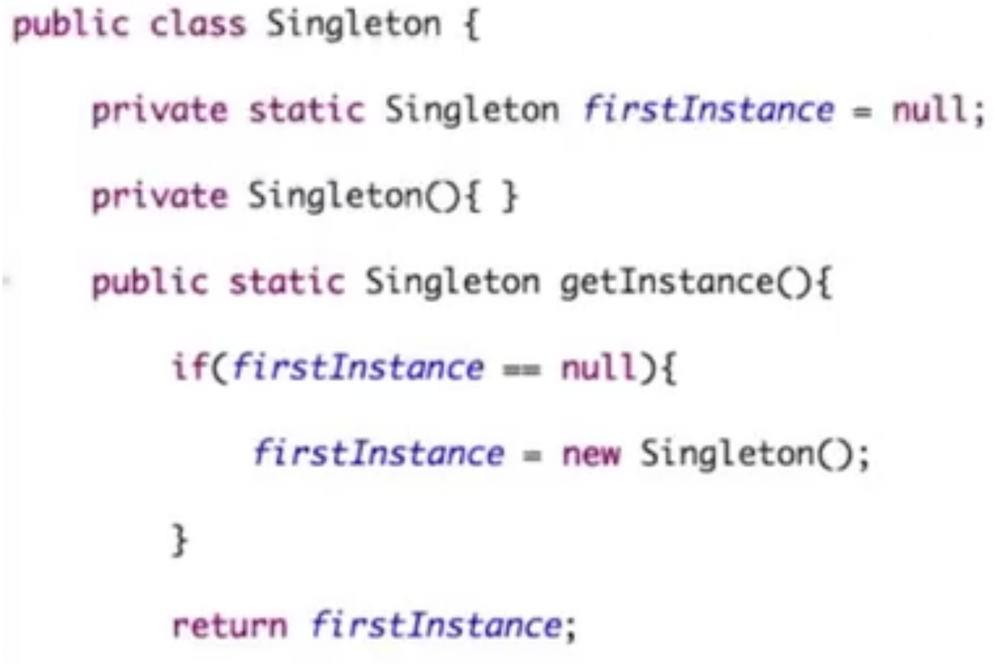
```
public class Product {
    public int Barcode;
}
```

Class variables:

These are also known as *static member variables* and there's only one copy of that variable that is shared with all instances of that class. If changes are made to that variable, all other instances will see the effect of the changes.

```
public class Product {
    public static int Barcode;
}
```

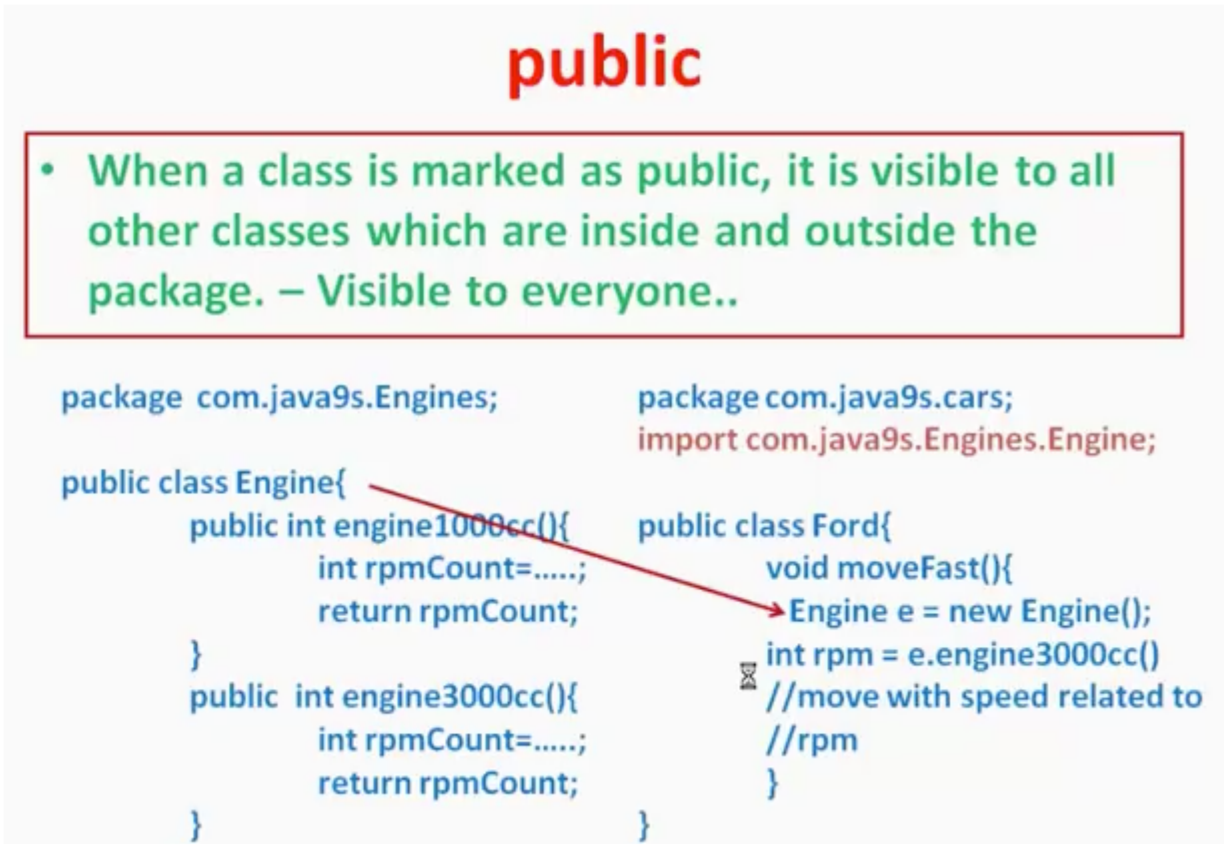
What is Singleton class —It is basically used when you want to eliminate the option of instantiating more than one object from the given class. Just one instance of that class allowed . So first instance variable below checks if instance already created or not.. if not it will allow only one instance to be created



```
public class Singleton {

    private static Singleton firstInstance = null;

    private Singleton(){ }

    public static Singleton getInstance(){

        if(firstInstance == null){

            firstInstance = new Singleton();

        }

        return firstInstance;
```

Constructor gets invoked when a new object is created. Every class has a constructor. If we do not explicitly write a constructor for a class the java compiler builds a default constructor for that class. An Object is first declared, then instantiated and then it is initialized. Default value of byte datatype is 0. The eight primitive types are byte, char, short, int, long, float, double, and boolean

What is **Access Modoifiers** —— It is important to retain confidentiality of certain algorithm written in the methods. Such logic can be protected from external sources.. so in another words, AM helps to implement the concept of **encapsulation** in OOPs. So it uses PRIVATE, PROTECTED, PUBLIC and DEFAULT keywords . AM helps to hide logic and data . AM can be at class level or or member level (e.g. method or instance variables)

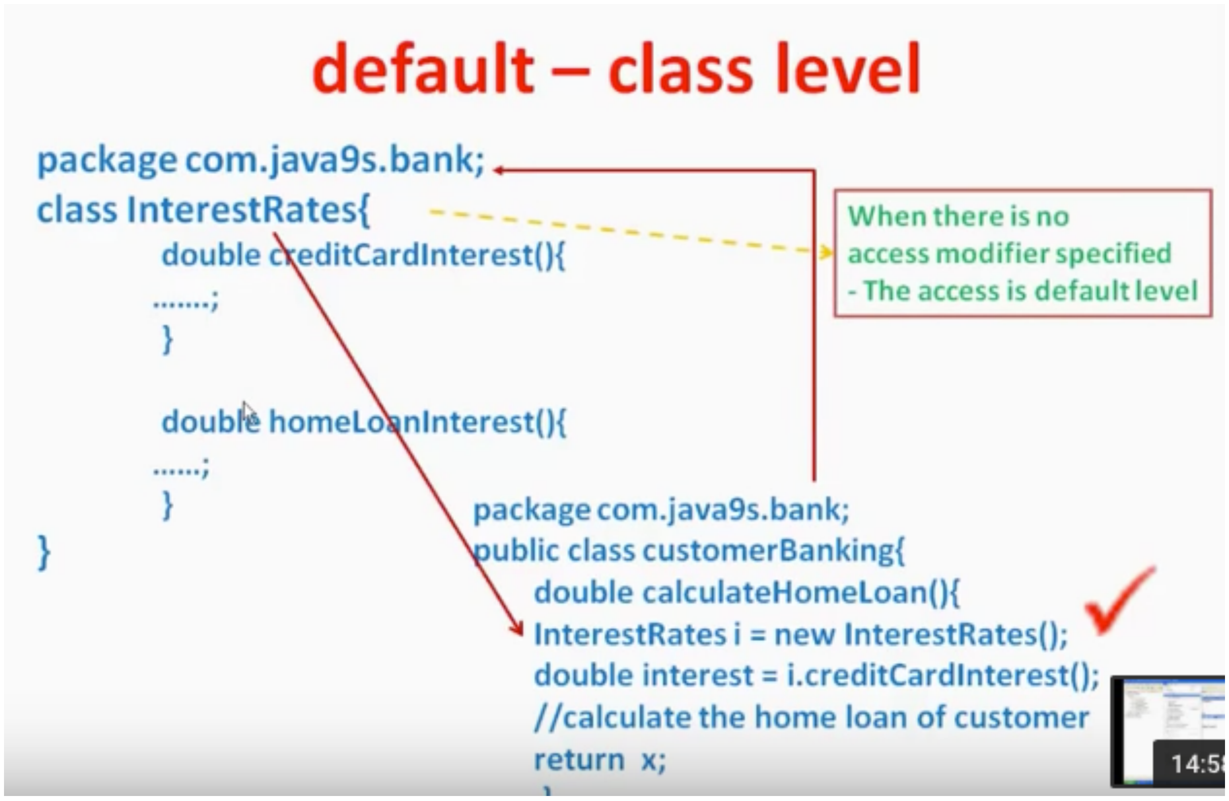


public

- When a class is marked as public, it is visible to all other classes which are inside and outside the package. – Visible to everyone..

```
package  com.java9s.Engines;

public class Engine{
          public int engine1000cc(){
                    int rpmCount=.....;
                    return rpmCount;
          }
          public  int engine3000cc(){
                    int rpmCount=.....;
                    return rpmCount;
          }
```

```
package com.java9s.cars;
import com.java9s.Engines.Engine;

public class Ford{
          void moveFast(){
           Engine e = new Engine();
          int rpm = e.engine3000cc()
          //move with speed related to
          //rpm
          }
}
```

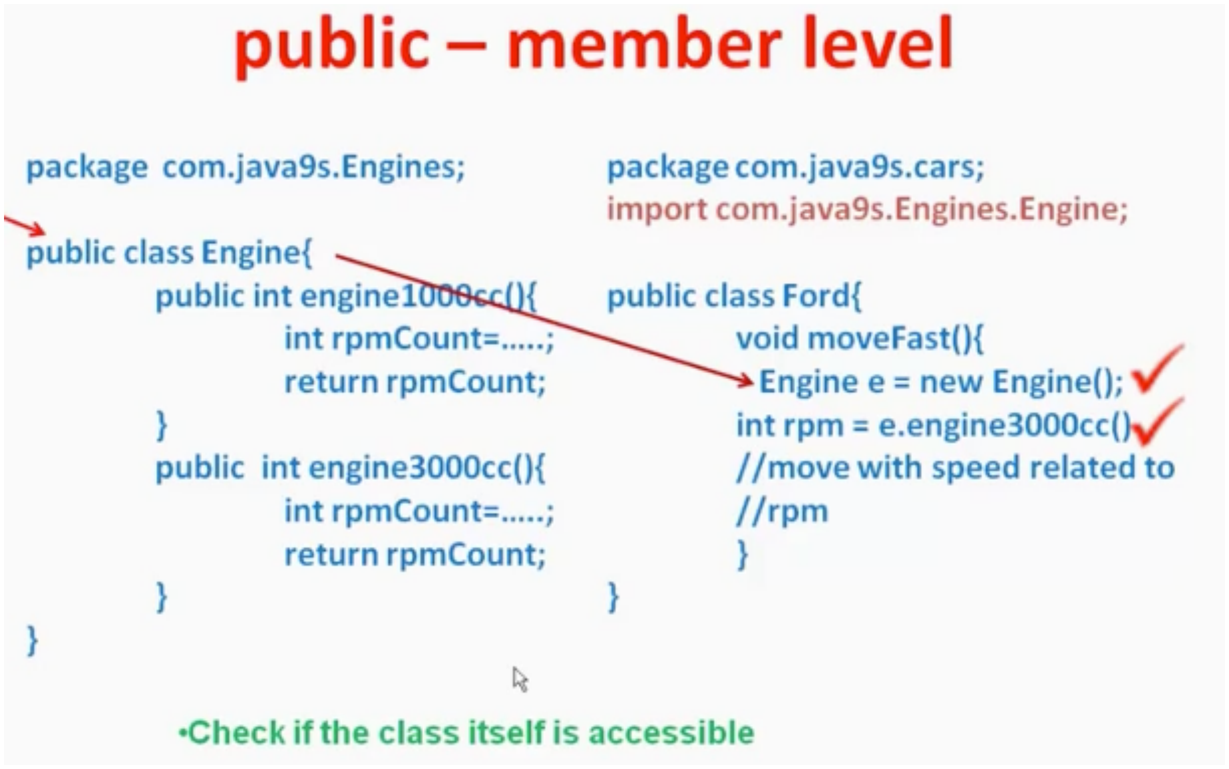


default – class level

```
package com.java9s.bank;
class InterestRates{
        double creditCardInterest(){
        .......;
        }

        double homeLoanInterest(){
        ......;
        }
}
```

When there is no access modifier specified - The access is default level

```
package com.java9s.bank;
public class customerBanking{
    double calculateHomeLoan(){
    InterestRates i = new InterestRates(); ✓
    double interest = i.creditCardInterest();
    //calculate the home loan of customer
    return x;
```

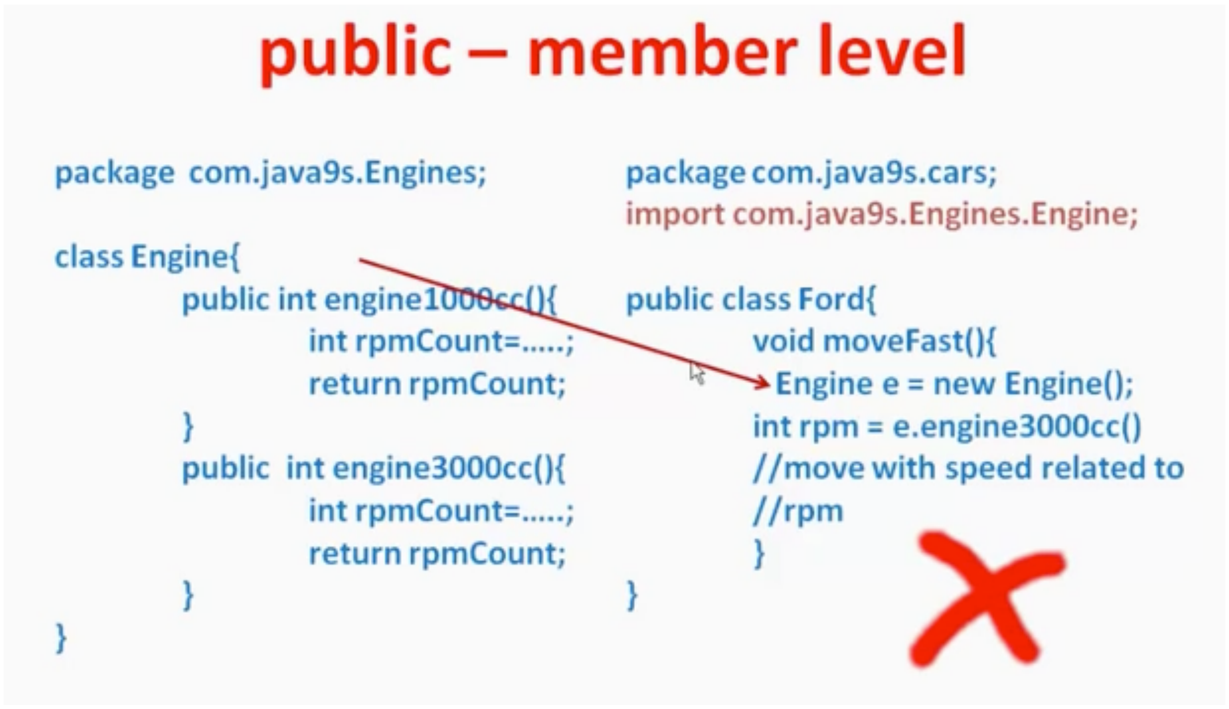


public – member level

```
package  com.java9s.Engines;

public class Engine{
        public int engine1000cc(){
                int rpmCount=.....;
                return rpmCount;
        }
        public  int engine3000cc(){
                int rpmCount=.....;
                return rpmCount;
        }
}
```

```
package com.java9s.cars;
import com.java9s.Engines.Engine;

public class Ford{
        void moveFast(){
        Engine e = new Engine(); ✓
        int rpm = e.engine3000cc() ✓
        //move with speed related to
        //rpm
        }
}
```

•Check if the class itself is accessible

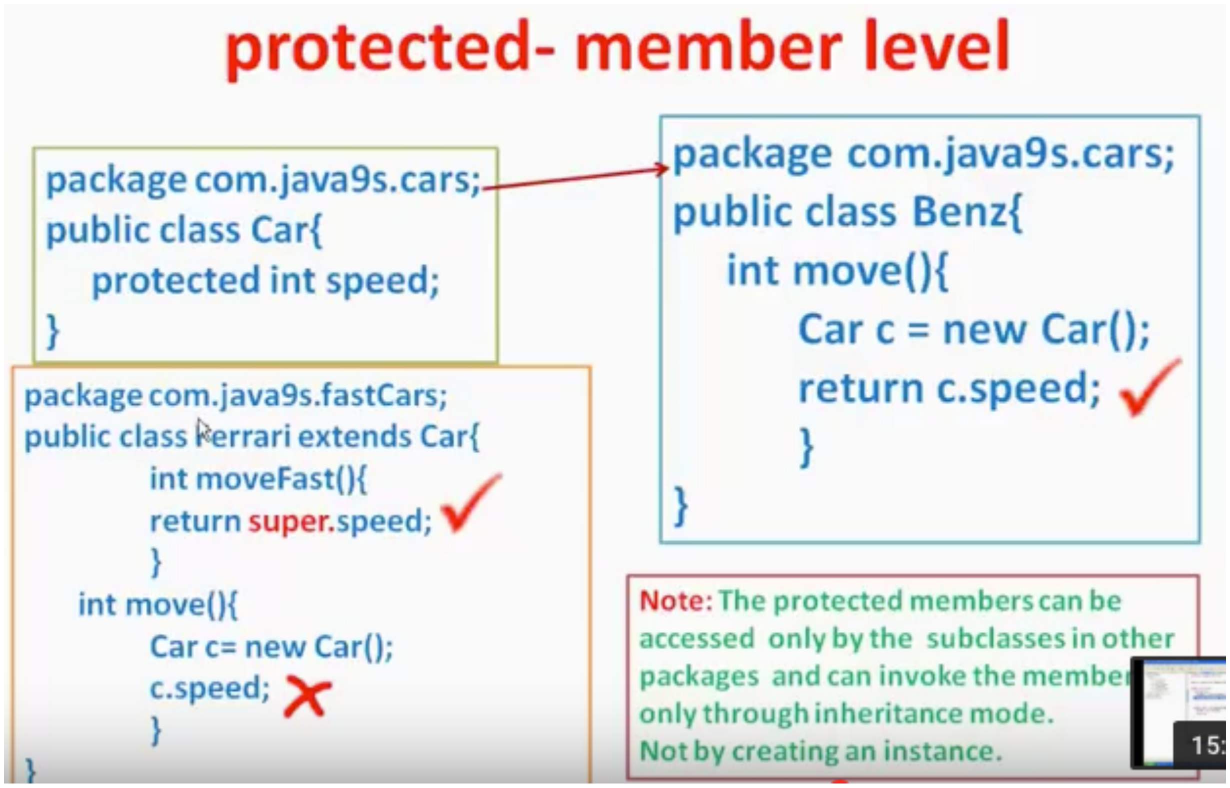


public – member level

```
package  com.java9s.Engines;

class Engine{
        public int engine1000cc(){
                int rpmCount=.....;
                return rpmCount;
        }
        public  int engine3000cc(){
                int rpmCount=.....;
                return rpmCount;
        }
}
```

```
package com.java9s.cars;
import com.java9s.Engines.Engine;

public class Ford{
        void moveFast(){
        Engine e = new Engine();
        int rpm = e.engine3000cc()
        //move with speed related to
        //rpm
        }
}
```

In case of protected - there has to be inheritance relationship between main class and subclass. Protected access modifier is applicable only to method and not at class level. So method can be accessible only when we use inheritance .. We cannot use inheritance at higher level like at class level … There is no higher level at make contract above class … so class never becomes protected modifiers or protected is not a legal access modifier for class



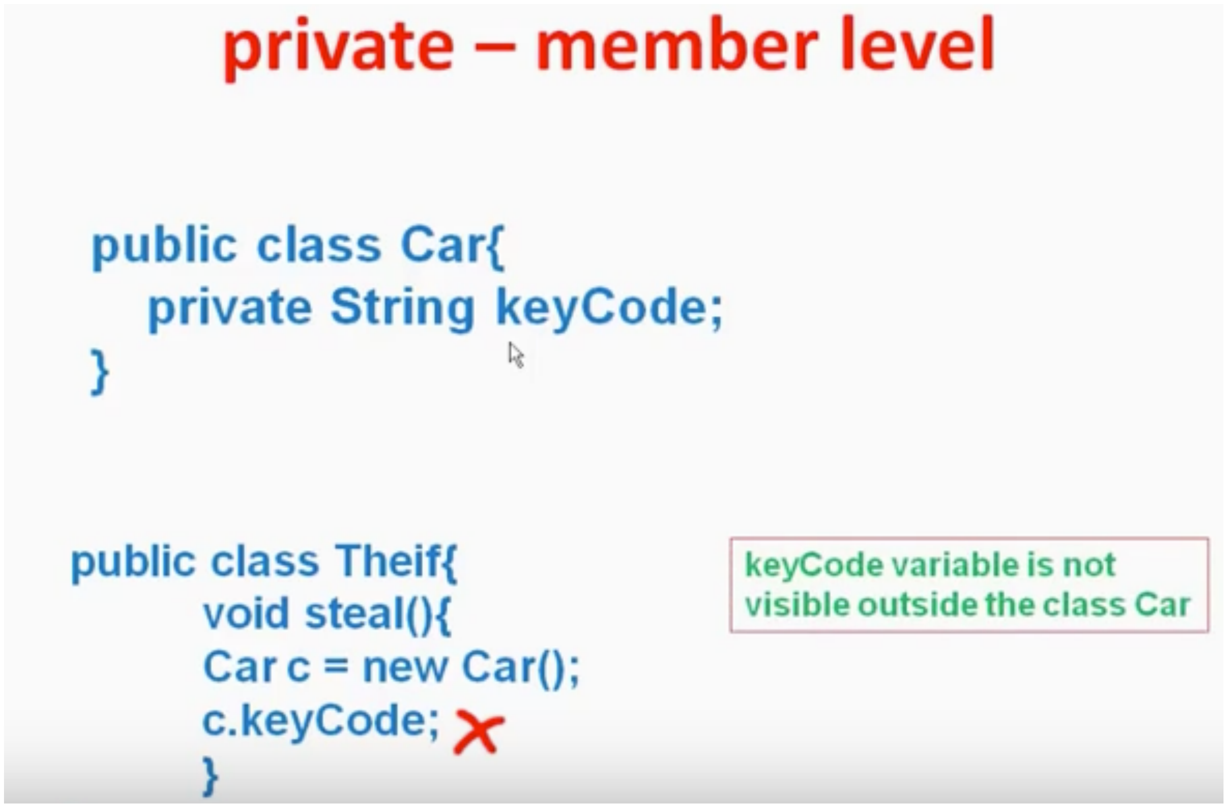
protected- member level

```
package com.java9s.cars;
public class Car{
    protected int speed;
}
```

```
package com.java9s.cars;
public class Benz{
    int move(){
        Car c = new Car();
        return c.speed; ✓
        }
}
```

```
package com.java9s.fastCars;
public class Ferrari extends Car{
        int moveFast(){
        return super.speed; ✓
        }
    int move(){
        Car c= new Car();
        c.speed; ✗
        }
}
```

Note: The protected members can be accessed only by the subclasses in other packages and can invoke the member only through inheritance mode. Not by creating an instance.



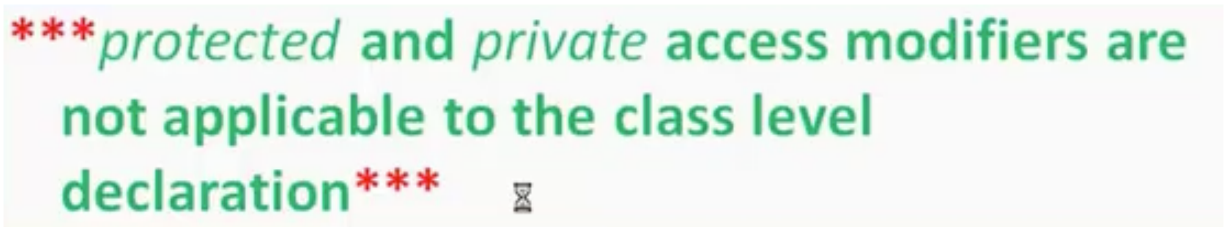
private – member level

```
public class Car{
    private String keyCode;
}
```

```
public class Theif{
        void steal(){
        Car c = new Car();
        c.keyCode; ✗
        }
```

keyCode variable is not visible outside the class Car

Protected and private is not applicable at the class level

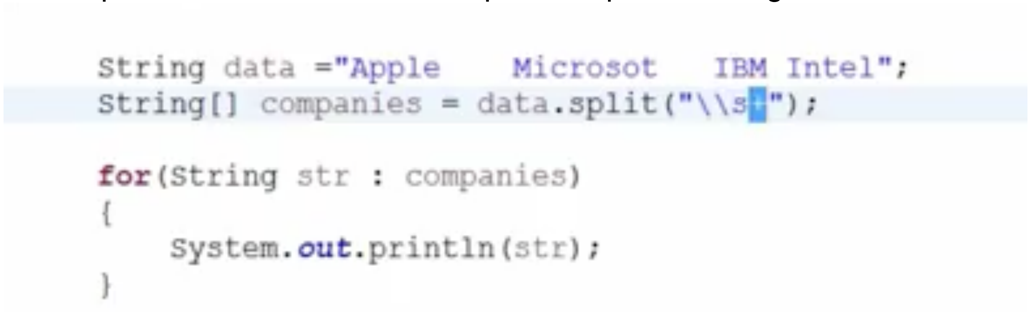

protected and private access modifiers are not applicable to the class level declaration

Access modifier should be applied to local variables of method

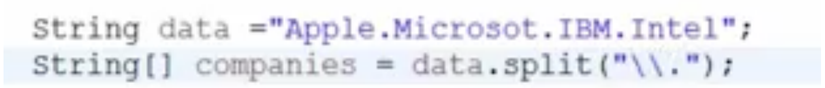
According to Java Operator precedence, which operator is considered to be with highest precedence? — Postfix operators i.e () [] . is at the highest precedence.
When parseInt() method can be used? —— This method is used to get the primitive data type of a certain String.

The String class is immutable, so that once it is created a String object cannot be changed. Since String is immutable it can safely be shared between many threads ,which is considered very important for multithreaded programming.If there is a necessity to make a lot of modifications to Strings of characters then StringBuffer should be used. Use StringBuilder whenever possible because it is faster than StringBuffer. But, if thread safety is necessary then use StringBuffer objects. A dead thread cannot be restarted.

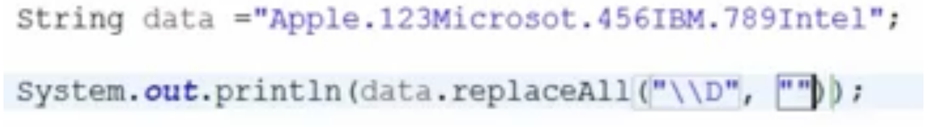
What is regex package in java ——— Basically regrex used for dealing with special way of handling text as shown in example below, we are using \\s+ which means multiple white space needs to be identified prior to split the string.


```
String data ="Apple    Microsot   IBM Intel";
String[] companies = data.split("\\s+");

for(String str : companies)
{
    System.out.println(str);
}
```

Another example, lets say we have dot as a separator -


```
String data ="Apple.Microsot.IBM.Intel";
String[] companies = data.split("\\.");
```

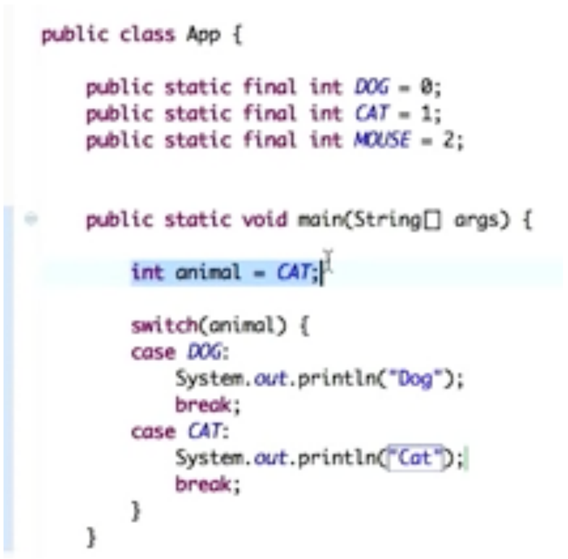
here is another example to replace all characters with spaces and retain only numerics


```
String data ="Apple.123Microsot.456IBM.789Intel";

System.out.println(data.replaceAll("\\D", ""));
```

java.util.regex consists of three classes – Pattern class, Matcher class and PatternSyntaxException class.

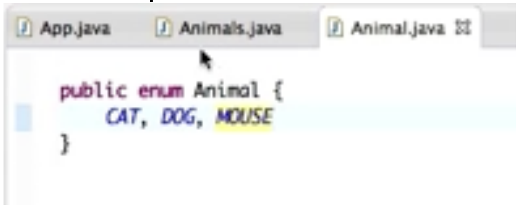
It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called finalize( ), and it can be used to ensure that an object terminates cleanly.

What is enum or Enumeration (**it is like a check table**) — As shown below, we have defined 3 variables for Dog, cat , mouse .. so we don't have to define these variables or set of values like this.. it can be done differently by creating enum



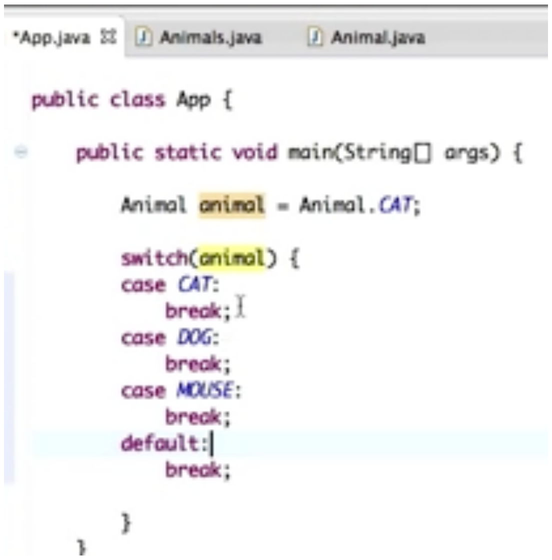
```
public class App {

    public static final int DOG = 0;
    public static final int CAT = 1;
    public static final int MOUSE = 2;


    public static void main(String[] args) {

        int animal = CAT;

        switch(animal) {
        case DOG:
            System.out.println("Dog");
            break;
        case CAT:
            System.out.println("Cat");
            break;
        }
    }
```

Now if we create Enum (enum stands for enumerated) class then it can store set of these values which can use in other class. So we don't need those 3 static variables defined in previous class above.



```
public enum Animal {
    CAT, DOG, MOUSE
}
```

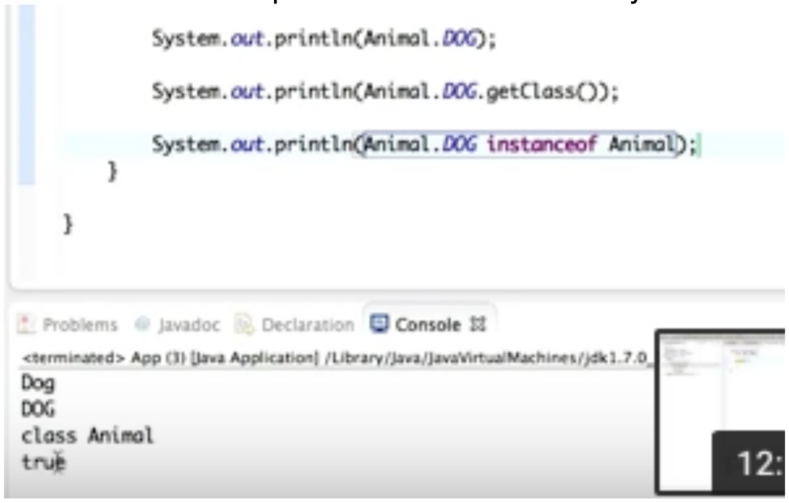
and now, we can refer this Enum Animal into our app class



*App.java Animals.java Animal.java

```
public class App {

    public static void main(String[] args) {

        Animal animal = Animal.CAT;

        switch(animal) {
        case CAT:
            break;
        case DOG:
            break;
        case MOUSE:
            break;
        default:
            break;

        }
    }
```

Here interesting thing is, all these set of values mentioned in enum are objects of Animal …when we print InstanceOf then it says true

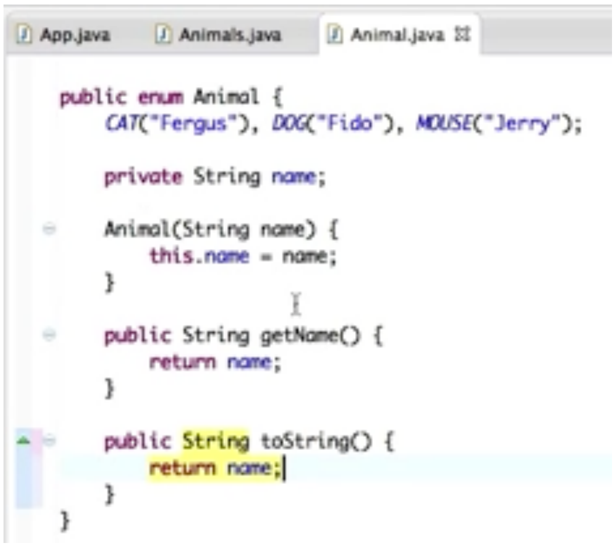


```
        System.out.println(Animal.DOG);

        System.out.println(Animal.DOG.getClass());

        System.out.println(Animal.DOG instanceof Animal);
    }

}
```

Problems Javadoc Declaration Console

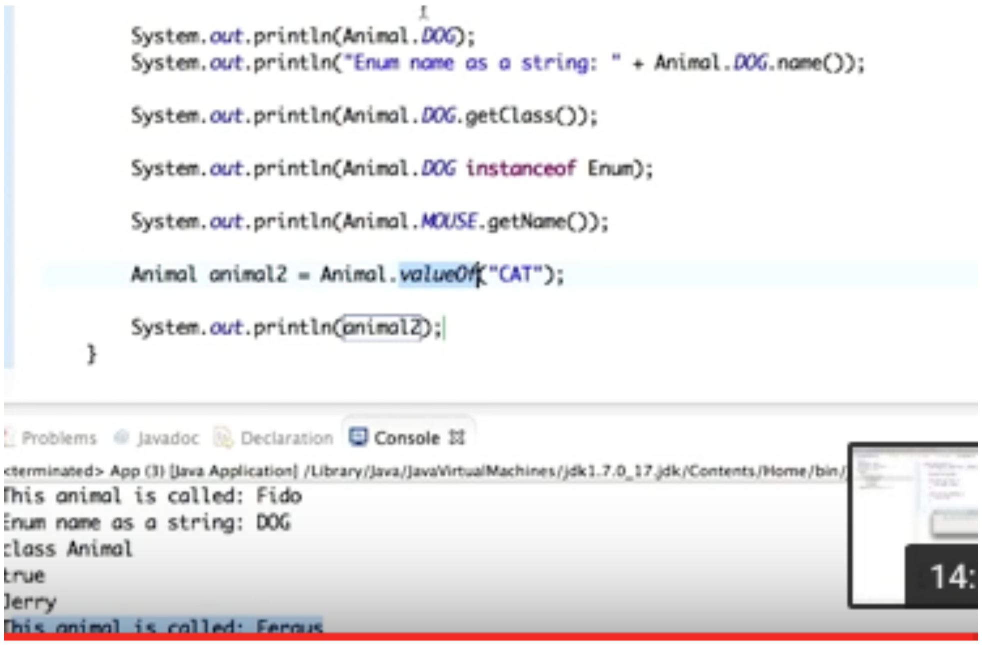
```
<terminated> App (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_
Dog
DOG
class Animal
true
```

Some of the advance topic for Enum —— It is also possible to declare construction; getter and setter , toString for enum as shown below



```
App.java    Animals.java    Animal.java

public enum Animal {
    CAT("Fergus"), DOG("Fido"), MOUSE("Jerry");

    private String name;

    Animal(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return name;
    }
}
```

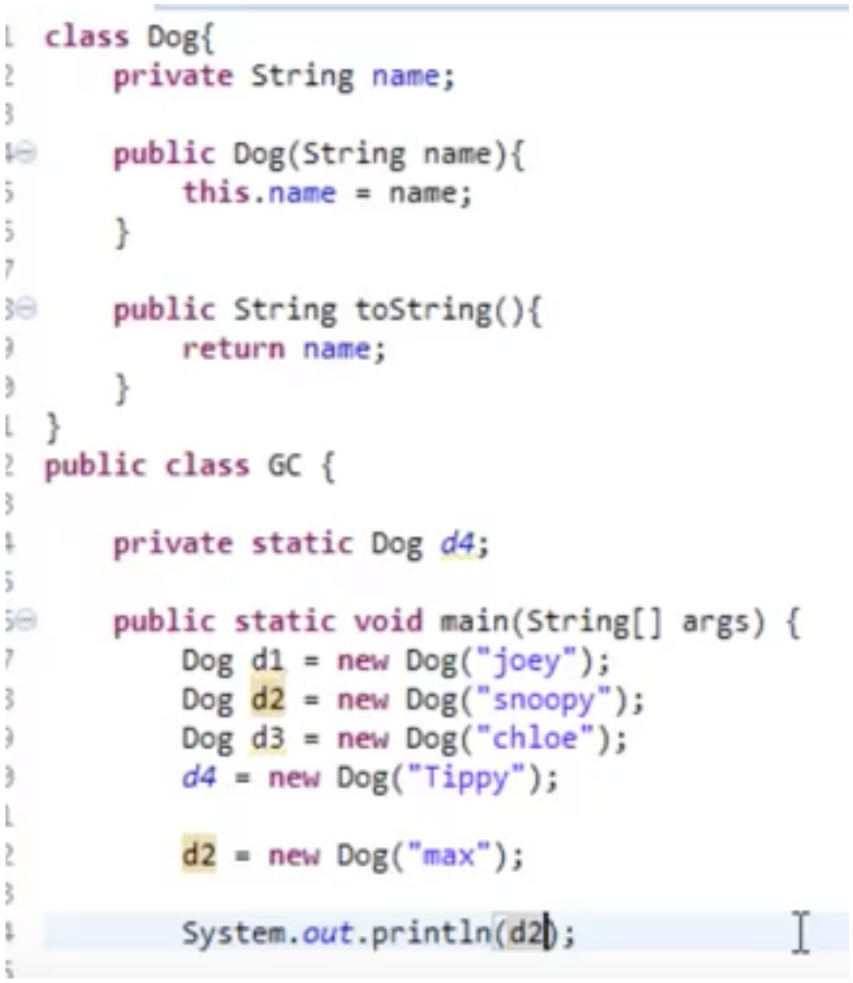
If we now call this enum from main class and print values.. as shown below



```
        System.out.println(Animal.DOG);
        System.out.println("Enum name as a string: " + Animal.DOG.name());

        System.out.println(Animal.DOG.getClass());

        System.out.println(Animal.DOG instanceof Enum);

        System.out.println(Animal.MOUSE.getName());

        Animal animal2 = Animal.valueOf("CAT");

        System.out.println(animal2);
    }
```

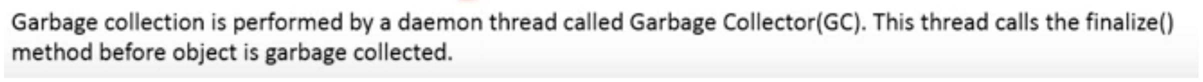
```
<terminated> App (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_17.jdk/Contents/Home/bin/
This animal is called: Fido
Enum name as a string: DOG
class Animal
true
Jerry
This animal is called: Fergus
```

What is Garbage Collection & Finalize() method does? —— When java program start in JVM , it gets some memory from OS .. JVM uses part of this memory
 to store objects created by the program , called heap memory. So when it gets full, we get OutOfMemory exception. GC is automatic memory management system that allocate memory when we get rid reference of some object that we created .. As shown below, when we instantiate new class like d1,d2,d3 etc, memory is allocated by java and it gets cleanup in example of d2 when we assign new instance to it ..example from “snoopy” to “max”.. This happens automatically.. This can also be done by methods…Pl note, this is applicable to object that are created by NEW keyword… for all other objects,. we can use finalize method…. which is described later in this topic.



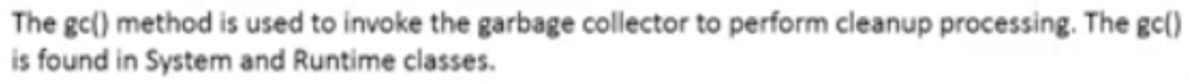
```
class Dog{
    private String name;

    public Dog(String name){
        this.name = name;
    }

    public String toString(){
        return name;
    }
}
public class GC {

    private static Dog d4;

    public static void main(String[] args) {
        Dog d1 = new Dog("joey");
        Dog d2 = new Dog("snoopy");
        Dog d3 = new Dog("chloe");
        d4 = new Dog("Tippy");

        d2 = new Dog("max");

        System.out.println(d2);
```

In example above, when GC figures out old value of d2 is not in use, GC will call finalize() method to perform clean up operation. Syntax of finalize() is as follows



Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

protected static finalize() {
}



The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

so when we call system.gc() , JVM will call GC dameon to perform clean up.. so it will call finalize() method before clean up or GC process takes place.. so logic written in finalize() method is final chance prior to GC takes place. Garbage collection does not guarantee that a program will not run out of memory.

What is checked exception —— It is an exception that is typically a **user error** or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs.

What is runtime exception —- It is an exception that occurs that probably could have been avoided by the **programmer**. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation. The Exception class has two main subclasses : IOException class and RuntimeException Class.

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred. The = operator is right associative.

What is **SUPER** keyword — If the method overrides one of its superclass's methods, overridden method can be invoked through the use of the keyword super. It can be also used to refer to a hidden field.

Polymorphism — It allows code interchangeability between interface and class . The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. In other words, polymorphism **allows you define one interface and have multiple implementations.**

What is abstract class —— A Java abstract class is a class which cannot be instantiated, meaning you cannot create new instances of an abstract class. The purpose of an abstract class is to function as a base for subclasses.

What is abstract method —— If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract. An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.Interface cannot be instantiated. An interface does not contain any constructors. All of the methods in an interface are abstract.

What is encapsulation — It is way to hide access to data with the help of access modifiers like protected, private , public and default

What is Multi-threading —— A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thread can be created by: implementing Runnable interface, extending the Thread class.

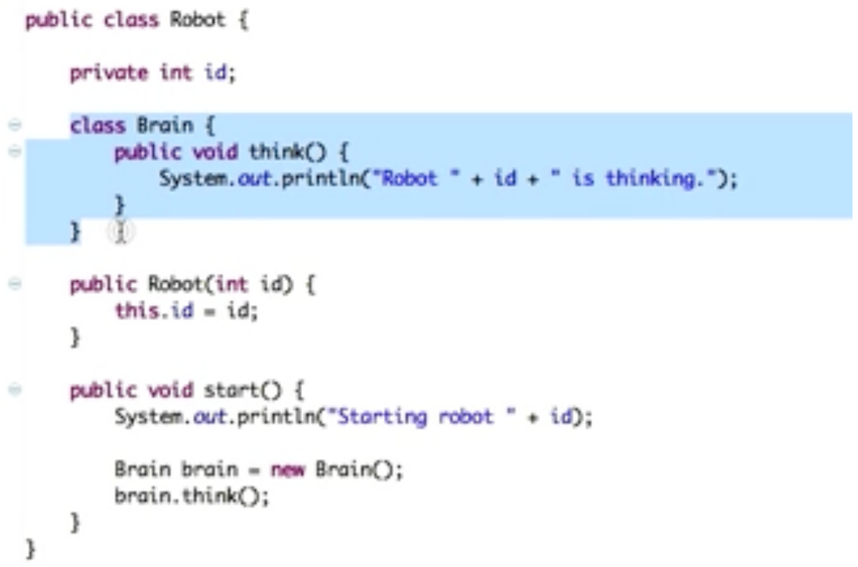
What is Applet —— An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal . An applet extends java.applet.Applet class.

What is this() — It is used with variables or methods and used to call constructer of same class. Java Runtime Environment is an implementation of the Java Virtual Machine which executes Java programs. It provides the minimum requirements for executing a Java application;

What is SET Interface —— It is a collection of element which cannot contain duplicate elements. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. Treeset is implemented when we want elements of the set in the sorted order . Comparable interface is used to sort collections and arrays of objects using the collections.sort() and java.utils.

What is **static** in public static void main (String args[ ]) – it allows main() to be called without instantiating a particular instance of a class.
JAR files is Java Archive fles and it aggregates many files into one. It holds Java classes in a library. J
WAR file — This is Web Archive File and used to store XML, java classes, and JavaServer pages. which is used to distribute a collection of JavaServer Pages, Java Servlets, Java classes, XML files, static Web pages etc.

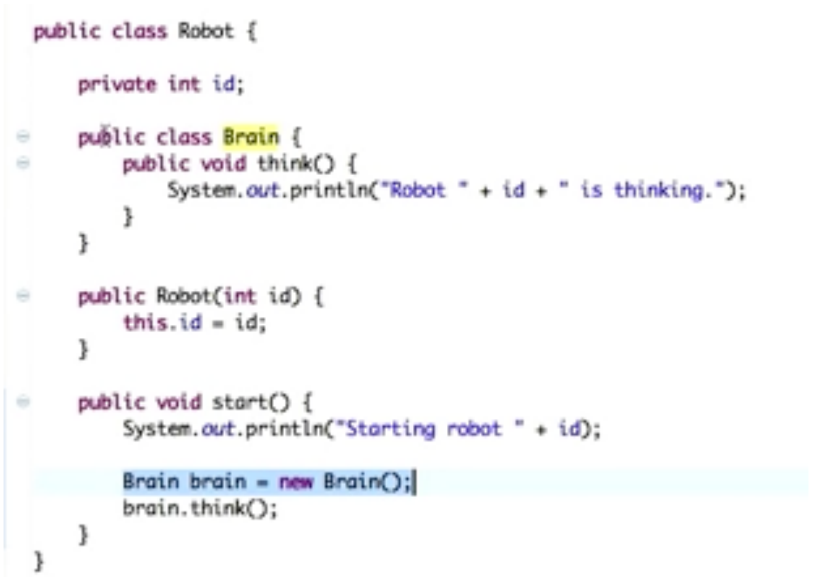
What is Nested class —Class defined within public class can access instance variable. as shown below .. example id. So here Brain is the nested class.



```
public class Robot {

    private int id;

    class Brain {
        public void think() {
            System.out.println("Robot " + id + " is thinking.");
        }
    }

    public Robot(int id) {
        this.id = id;
    }

    public void start() {
        System.out.println("Starting robot " + id);

        Brain brain = new Brain();
        brain.think();
    }
}
```

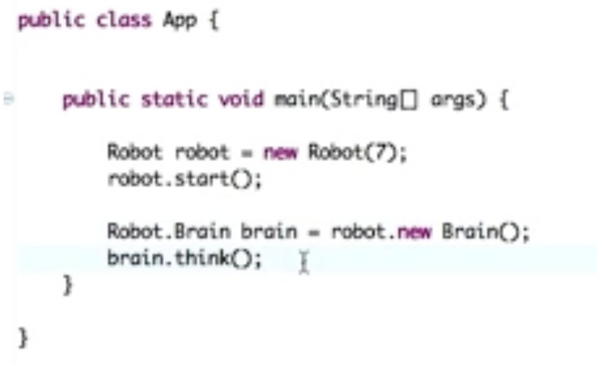
So it also possible to use various access modifier to this Brain class . such as private ….

What is Inner class —
First type is — This is like accessing subclass of other class with first instantiating main class and then subclass… So syntax is bit different as shown below



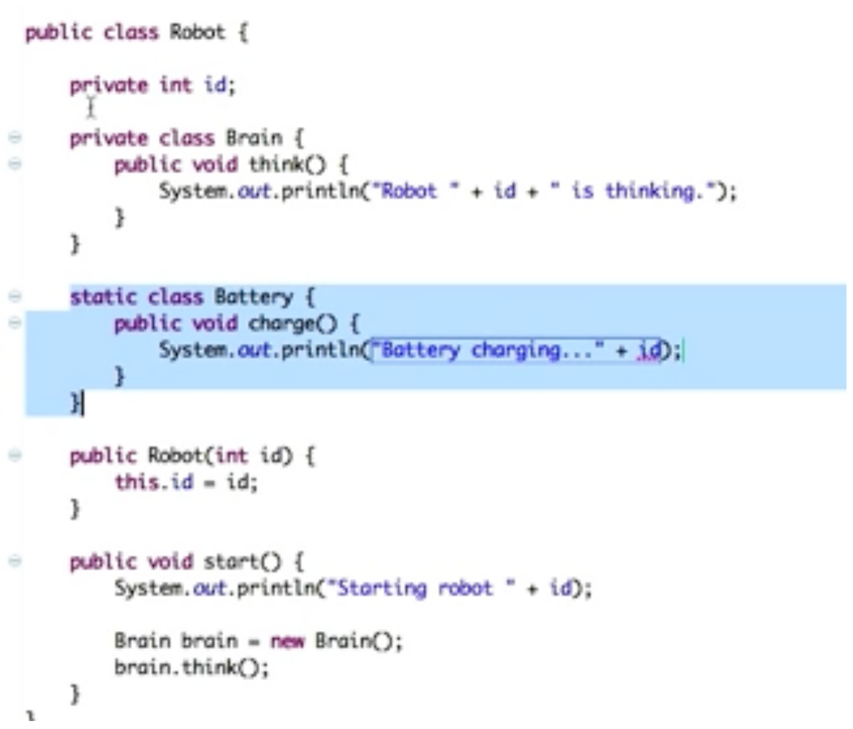
```
public class Robot {

    private int id;

    public class Brain {
        public void think() {
            System.out.println("Robot " + id + " is thinking.");
        }
    }

    public Robot(int id) {
        this.id = id;
    }

    public void start() {
        System.out.println("Starting robot " + id);

        Brain brain = new Brain();
        brain.think();
    }
}
```

as shown below, we are first instantiating robot class and then dot to instantiate Brain class. So Brain now becomes inner class. Brian class needs to be public , not private , else it will not work.



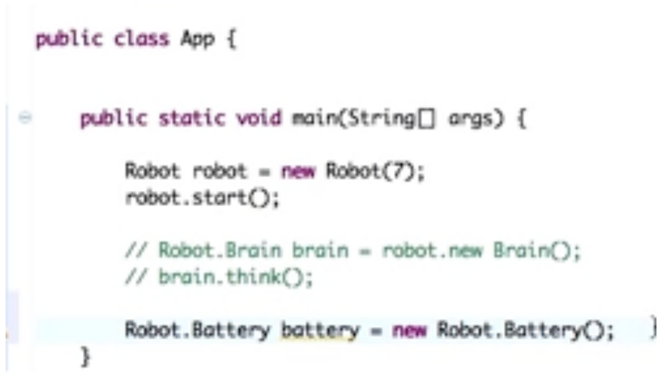
```
public class App {


    public static void main(String[] args) {

        Robot robot = new Robot(7);
        robot.start();

        Robot.Brain brain = robot.new Brain();
        brain.think();
    }

}
```

Second type of inner class — In the type, we defined class as static as shown below.. This class Battery cannot access instance variable like id.. it gives an error however this class can be accessed from other main class as shown in second screen shot



```java
public class Robot {

    private int id;

    private class Brain {
        public void think() {
            System.out.println("Robot " + id + " is thinking.");
        }
    }

    static class Battery {
        public void charge() {
            System.out.println("Battery charging..." + id);
        }
    }

    public Robot(int id) {
        this.id = id;
    }

    public void start() {
        System.out.println("Starting robot " + id);

        Brain brain = new Brain();
        brain.think();
    }
}
```

see how battery class is instantiated



```java
public class App {


    public static void main(String[] args) {

        Robot robot = new Robot(7);
        robot.start();

        // Robot.Brain brain = robot.new Brain();
        // brain.think();

        Robot.Battery battery = new Robot.Battery();
    }
```

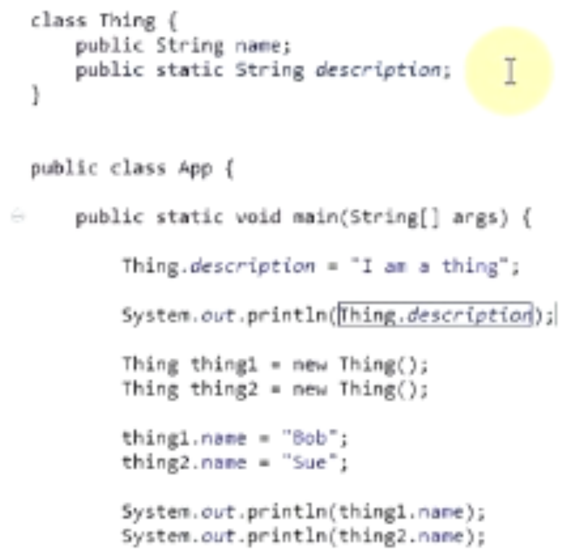
So basically non static inner class are used for grouping functionality and need access to instance variable of main class whereas static inner class are just normal class with no access to instance variables and can be used in other classes easily.

Order of precedence determines the order in which operators are evaluated in expressions. Associatity determines whether an expression is evaluated left-to-right or right-to-left.

What is Static and Final Keyword — So basically static variable are defined at class level and as shown below, name which is not static can be get it's own copy if we instantiate Thing class or create an object of it .. so we can have multiple copies of name however **description is static so we can get only one copy of it .. we cannot**

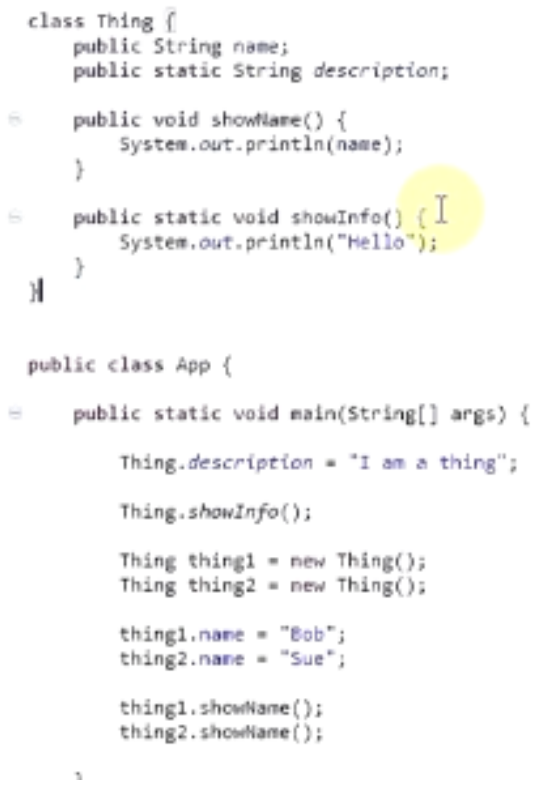
have multiple copies by instantiating class. So description is accessed by using class name directly. So if we do dropdown on thing1 or thing2 , we will not see description field…because it is static and not created for each new instance.

Example - Thread.sleep is static variable of Thread classes and if we define Thread T1 = new Thread then we can’t use T1.sleep because Sleep is static variable which is at class level (Thread)



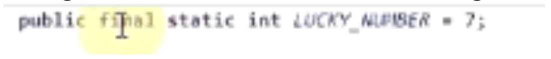
```
class Thing {
    public String name;
    public static String description;
}


public class App {

    public static void main(String[] args) {

        Thing.description = "I am a thing";

        System.out.println(Thing.description);

        Thing thing1 = new Thing();
        Thing thing2 = new Thing();

        thing1.name = "Bob";
        thing2.name = "Sue";

        System.out.println(thing1.name);
        System.out.println(thing2.name);
```

Now same applicable for methods.. As shown below, showman is not static method so each instance of class we create or object we create, we get new copy of that method however ShowInfo method is static so we can access it only at class level and each instance created for class Thing does not get copy of it .. Keep in mind that , static method cannot access instance variable which is not static …. so ShowInfo cannot access name since name is not static however instance method like ShowName can access description after thing1 or thing2 object is created…



```
class Thing {
    public String name;
    public static String description;

    public void showName() {
        System.out.println(name);
    }

    public static void showInfo() {
        System.out.println("Hello");
    }
}


public class App {

    public static void main(String[] args) {

        Thing.description = "I am a thing";

        Thing.showInfo();

        Thing thing1 = new Thing();
        Thing thing2 = new Thing();

        thing1.name = "Bob";
        thing2.name = "Sue";

        thing1.showName();
        thing2.showName();

    }
```

FINAL — This keyword is used to define constant variable . As shown below, once it is assigned value, it cannot be changed.



```
public final static int LUCKY_NUMBER = 7;
```

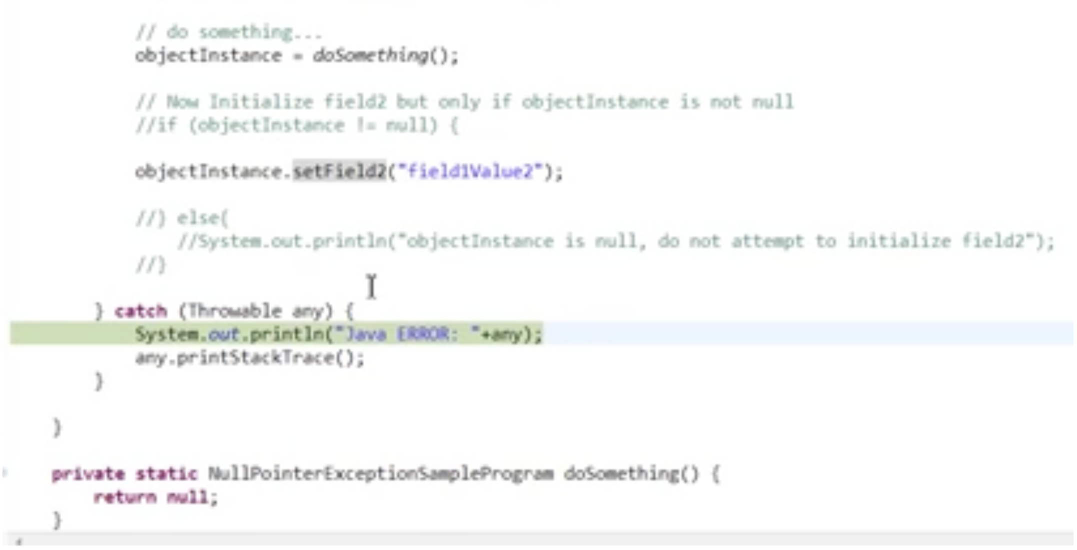
Down casting — It is the casting from a general to a more specific type, i.e. casting down the hierarchy.

JIT Complier — It improves the runtime performance of computer programs based on bytecode. The java compiler creates a default constructor only if there is no constructor in the class. Constructor cannot be made final. Constructor cannot be inherited.

Path and Classpath are operating system level environment variables. Path is defines where the system can find the executables(.exe) files and classpath is used to specify the location of .class files.
The size is the number of elements actually stored in the vector, while capacity is the maximum number of elements it can store at a given instance of time. Yes a Vector can contain heterogenous objects. Because a Vector stores everything in terms of Object. ArrayList can grow dynamically and provides more powerful insertion and search mechanisms than arrays.

What is function or method overloading — If a class has multiple functions by same name but different parameters or signature, it is known as Method **Overloading**. If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method override or **Overriding**. Final classes are created so the methods implemented by that class cannot be overridden. It can't be inherited. Overridden methods must have the same name, argument list, and return type. The overriding method may not limit the access of the method it overrides.

A NullPointerException (this is unchecked exception) is thrown when calling the instance method of a null object, accessing or modifying the field of a null object etc. As shown below in example, method at the bottom is passing null value to objectInstance and thus java throws an NullPointerException



```
            // do something...
            objectInstance = doSomething();

            // Now Initialize field2 but only if objectInstance is not null
            //if (objectInstance != null) {

            objectInstance.setField2("field1Value2");

            //} else{
                //System.out.println("objectInstance is null, do not attempt to initialize field2");
            //}

        } catch (Throwable any) {
            System.out.println("Java ERROR: "+any);
            any.printStackTrace();
        }

    }

    private static NullPointerExceptionSampleProgram doSomething() {
        return null;
    }
```

If System.exit (0); is written at the end of the try block, will the finally block still execute? — No in this case the finally block will not execute because when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

Double value can be cast to a byte. Constructor Chaining — A child object constructor always first needs to construct its parent. In Java it is done via an implicit call to the no-args constructor as the first statement.

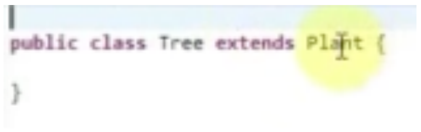
An object's lock is a mechanism that is used by multiple threads to obtain synchronized access to the object. A thread may execute a synchronized method of an object only after it has acquired the object's lock.

— Yes an Interface can inherit another Interface, for that matter an Interface can extend more than one Interface.

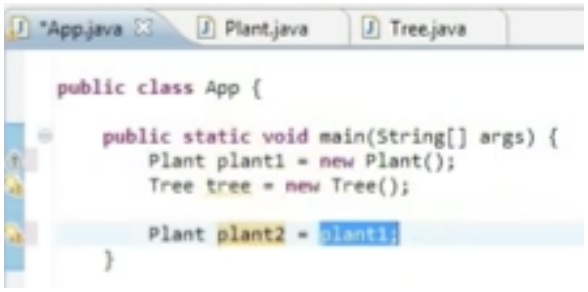
Which object oriented Concept is achieved by using overloading and overriding? —— Polymorphism

So basically what Polymorphism means — **if you have a child class of parent class**

then wherever you plan to use parent class, you can use child class there. In example below, tree is child of plant .. tree is extending plant class.. So in app.class , we can use tree class if we are using plant class.
In example below, plant2 is reference to plant object which is instantiated from plant class.. So since tree is extending plant , we can also use tree directly here …

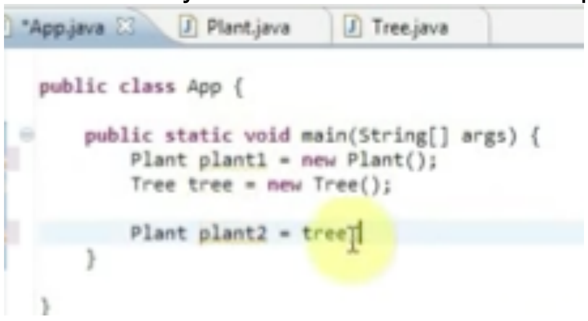


```
public class Tree extends Plant {

}
```



```
public class App {

    public static void main(String[] args) {
        Plant plant1 = new Plant();
        Tree tree = new Tree();

        Plant plant2 = plant1;
    }
```

So technically we can use tree instead of plant as shown below



```
public class App {

    public static void main(String[] args) {
        Plant plant1 = new Plant();
        Tree tree = new Tree();

        Plant plant2 = tree;
    }

}
```

here is video —very good — https://www.youtube.com/watch?v=daXvUxhBtAQ

Does it matter in what order catch statements for FileNotFoundException and IOException are written?   —— Yes, it does. The FileNoFoundException is inherited from the IOException. Exception's subclasses have to be caught first. The ArithmeticException is thrown when integer is divided by zero

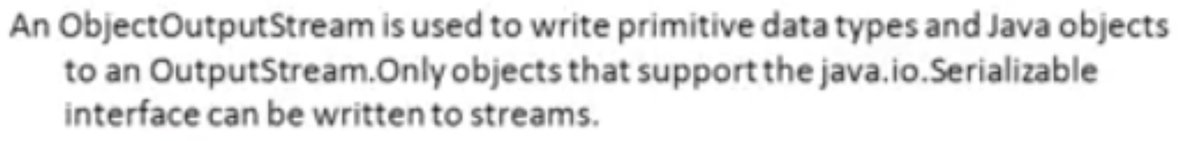
When a task invokes its yield() method, it returns to the ready state. When a task invokes its sleep() method, it returns to the waiting state.
The Vector class provides the capability to implement a growable array of objects. Vector proves to be very useful if you don't know the size of the array in advance, or you just need one that can change sizes over the lifetime of a program. The code sleep(2000); puts thread aside for exactly two seconds. The code wait(2000), causes a wait of up to two second.

Unicode - 16 bits ,

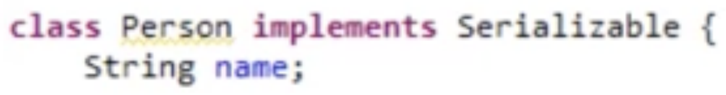
ASCII - 8 bits

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.
The Class class is used to obtain information about an object's design and java.lang.Class class instance represent classes, interfaces in a running Java application. A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

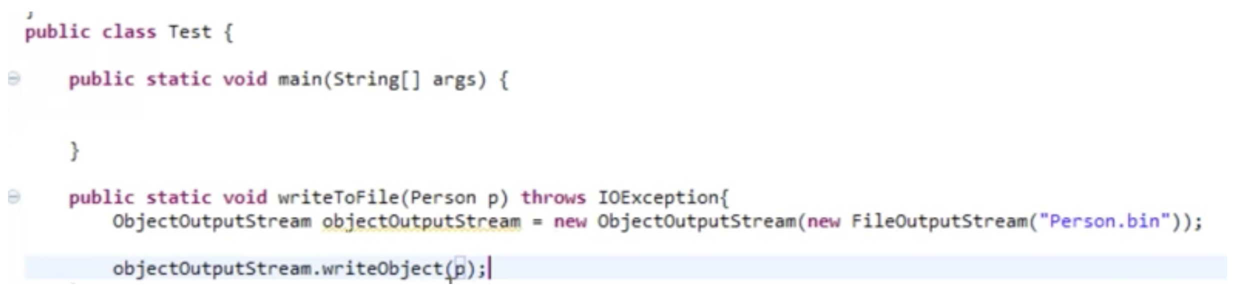
WHAT IS SERIALIZATION AND DESERIALIZATION ? —— Serialization is writing a state of an object into byte stream so object state can be transferred over the network .. Network takes only byte code … serialization interface helps write bytes to binary files for later use. So basically converting human readable object to binary format is serialization. A transient variable is a variable that may not be serialized during Serialization and which is initialized by its default value during de-serialization,


An ObjectOutputStream is used to write primitive data types and Java objects to an OutputStream.Only objects that support the java.io.Serializable interface can be written to streams.

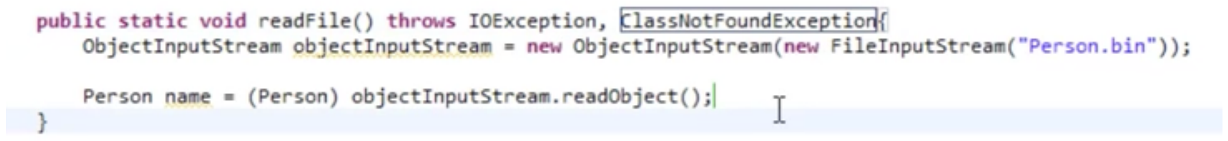
1. Extend class to implement serialization interface


```
class Person implements Serializable {
    String name;
```

2. Now this person class can be reference into another class and use below interfaces to write data to binary (.bin) file as shown below


```
public class Test {

    public static void main(String[] args) {


    }

    public static void writeToFile(Person p) throws IOException{
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream("Person.bin"));

        objectOutputStream.writeObject(p);
```

3. Once .bin file is generated, data can be read using cast operator as shown below


```
    public static void readFile() throws IOException, ClassNotFoundException{
        ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream("Person.bin"));

        Person name = (Person) objectInputStream.readObject();
    }
```

Can you write a Java class that could be used both as an applet as well as an application? — Yes, just add a main() method to the applet. AWT components are heavy-weight, whereas Swing components are lightweight. Heavy weight components depend on the local windowing toolkit. For example, java.awt.Button is a heavy weight component, when it is running on the Java platform for Unix platform, it maps to a real Motif button.

Constructors must have the same name as the class and can not return a value. They are only called once while regular methods could be called many times.
Can you call one constructor from another if a class has multiple constructors? —— Yes, use this() syntax. Default value of the boolean type is false.

A class loader is an object that is responsible for loading classes. The class ClassLoader is an abstract class. An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation. Default value of an object reference declared as an instance variable? —NULL

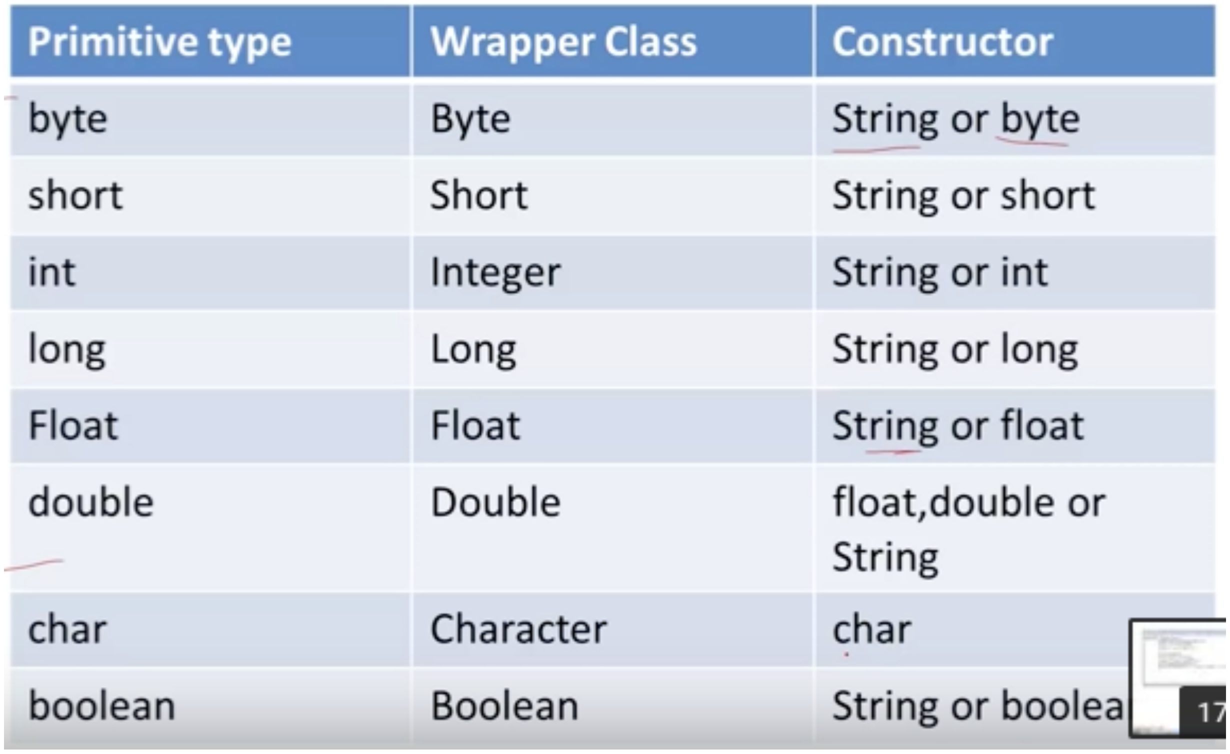
The Frame class extends Window to define a main application window that can have a menu bar. javax.Swing package. All components in Swing, except JApplet, JDialog, JFrame and JWindow are lightweight components.

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block or a finally block.

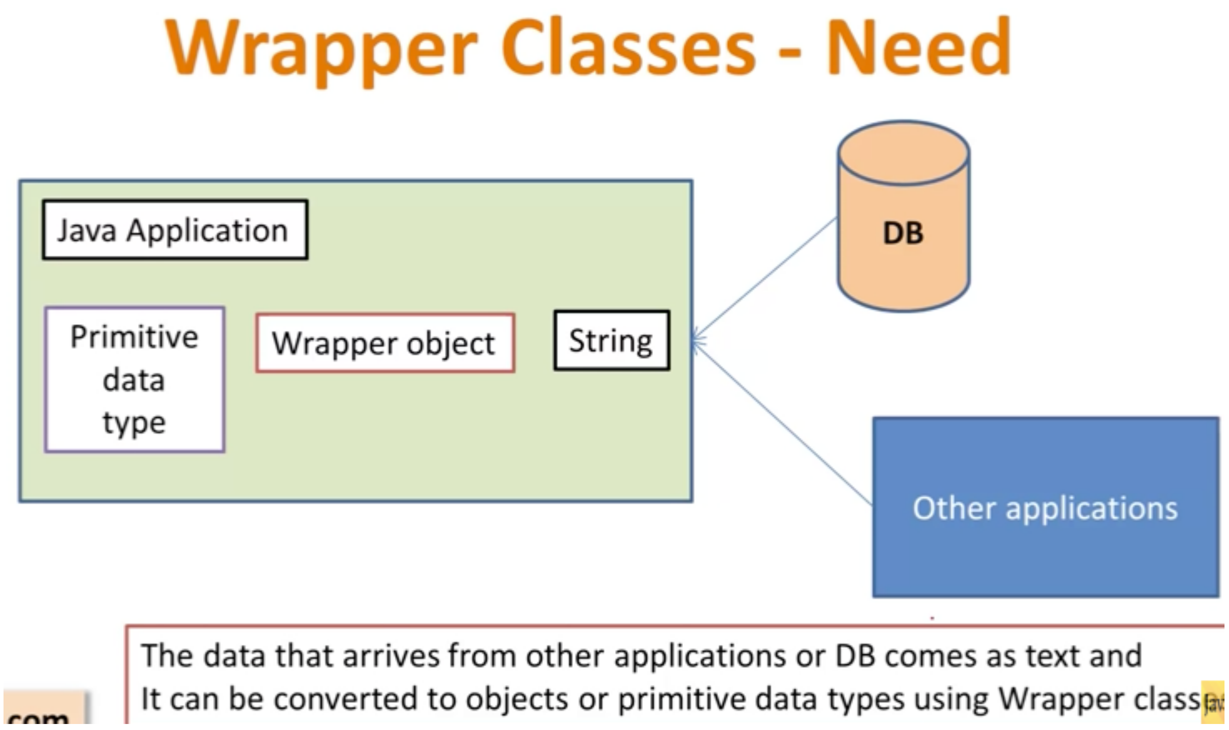
Wrapper classes (WC) —— WC helps in wrapping primitive data types into an object . These are classes that allow primitive types to be accessed as objects. Example: Integer, Character, Double, Boolean etc. So basically primitive data types in Java are not objects , like in C, these are implemented as data types therefore java is not 100% OOPS .. WC can be helpful to convert one type to another -

1. From primitives to string
2. primitive to another primitives
3. Binay , Octa,Hexa etc

WC are mainly used to store data in collection .



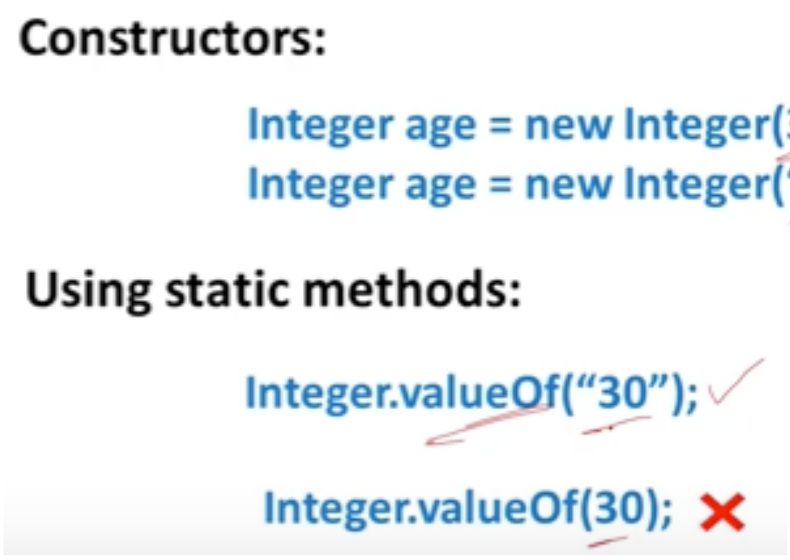
Primitive type	Wrapper Class	Constructor
byte	Byte	String or byte
short	Short	String or short
int	Integer	String or int
long	Long	String or long
Float	Float	String or float
double	Double	float,double or String
char	Character	char
boolean	Boolean	String or boolea



Wrapper Classes - Need

The data that arrives from other applications or DB comes as text and
It can be converted to objects or primitive data types using Wrapper class

Two ways to create wrapper object as follows



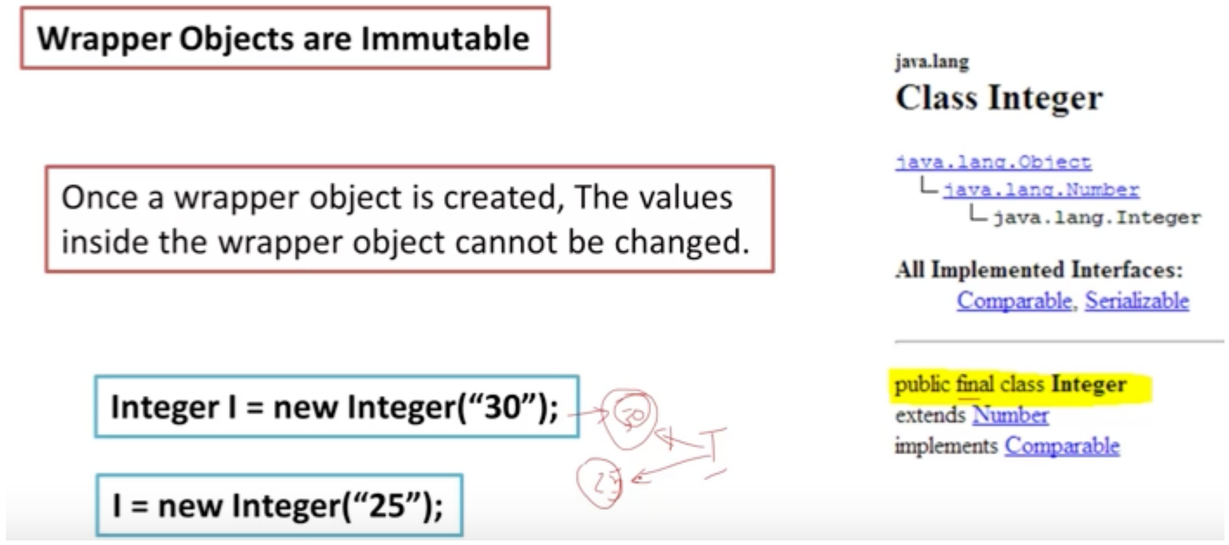
Constructors:

```
Integer age = new Integer(30);
Integer age = new Integer("30");
```

Using static methods:

```
Integer.valueOf("30"); ✓

Integer.valueOf(30); ✗
```

Since wrapper object are non-changeable , we use final keyword when we define class.



Wrapper Objects are Immutable

Once a wrapper object is created, The values inside the wrapper object cannot be changed.

```
Integer I = new Integer("30");

I = new Integer("25");
```

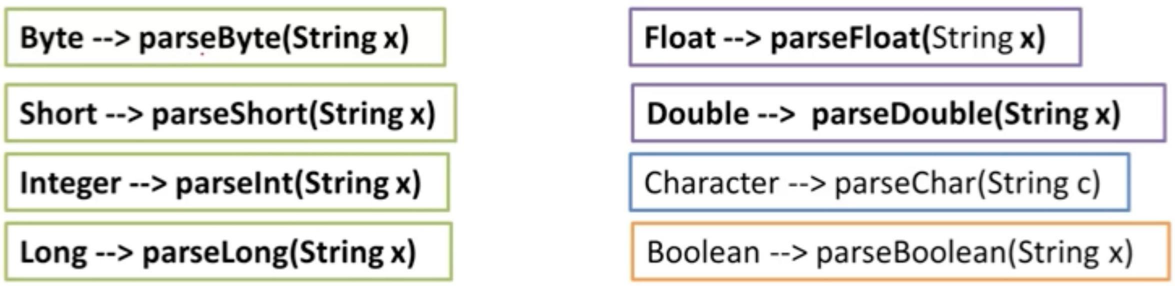
java.lang
Class Integer

```
java.lang.Object
  └ java.lang.Number
      └ java.lang.Integer
```

All Implemented Interfaces:
Comparable, Serializable

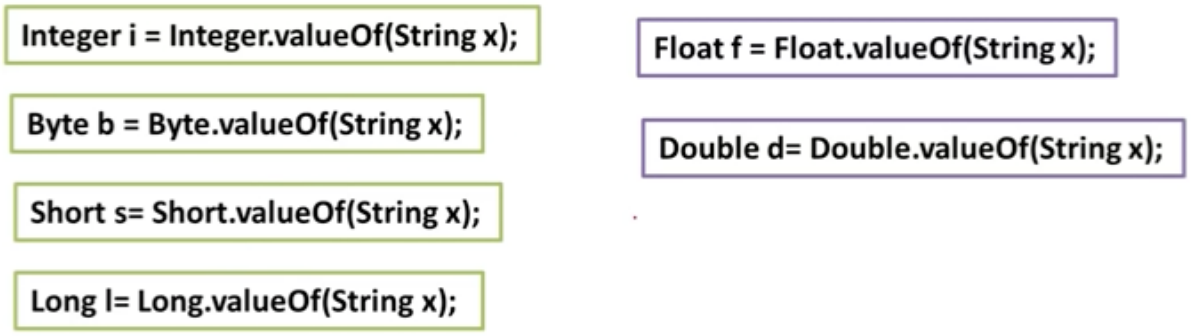
public final class **Integer**
extends Number
implements Comparable

Here is parseXX method that is used for conversion shown above in the table



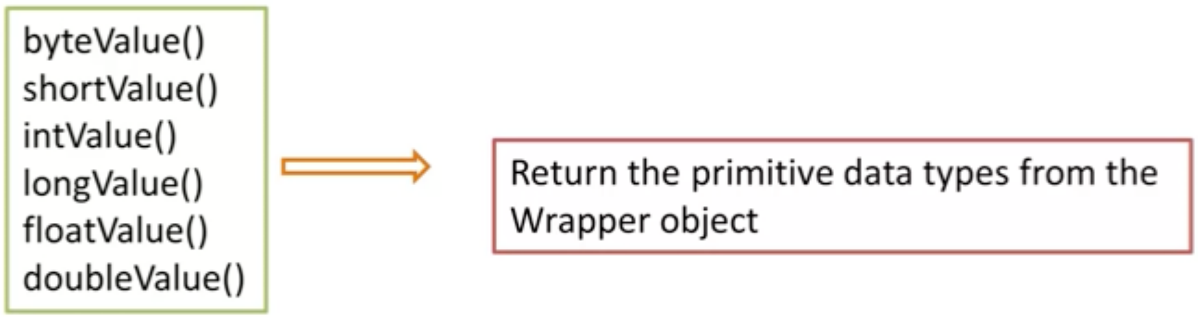
Byte --> parseByte(String x)	Float --> parseFloat(String x)
Short --> parseShort(String x)	Double --> parseDouble(String x)
Integer --> parseInt(String x)	Character --> parseChar(String c)
Long --> parseLong(String x)	Boolean --> parseBoolean(String x)

And now valueOf() will give us wrapper object as shown below



```
Integer i = Integer.valueOf(String x);
Byte b = Byte.valueOf(String x);
Short s= Short.valueOf(String x);
Long l= Long.valueOf(String x);
Float f = Float.valueOf(String x);
Double d= Double.valueOf(String x);
```

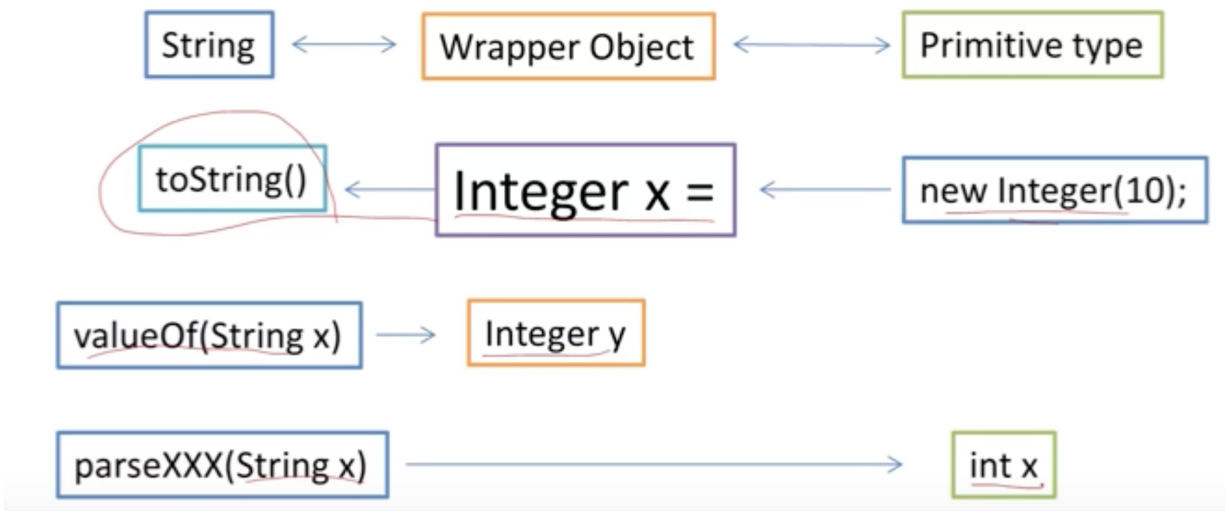
So basically , valueOf(String x) is wrapper classes helps to convert a string and returns the wrapper object . Once we got the wrapper object , we then use below value() method to convert wrapper object into primitive data type



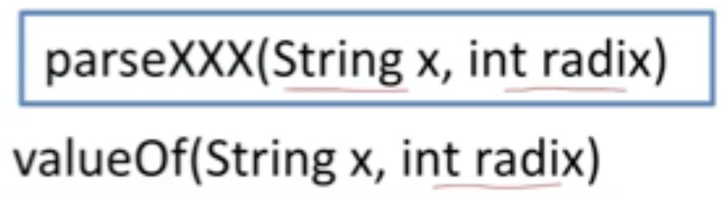
```
byteValue()
shortValue()
intValue()
longValue()
floatValue()
doubleValue()
```

Return the primitive data types from the Wrapper object

So summary is as follows -



In some cases, if string is passed in different format then radix argument can be used as shown below



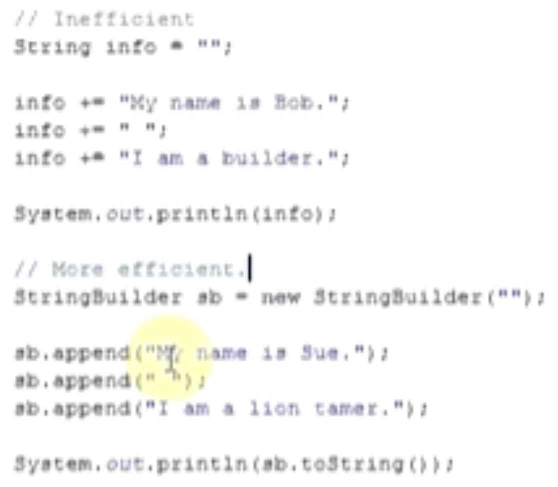
parseXXX(String x, int radix)

valueOf(String x, int radix)

Java Cache tools and methods -
- Java Caching System (JCS)
- OSCache
- Java Object Cache (JOCache)
- Java Caching Service, an open source implementation of the JCache API (SourceForge.net)
- SwarmCache
- JBossCache
- IronEye Cache

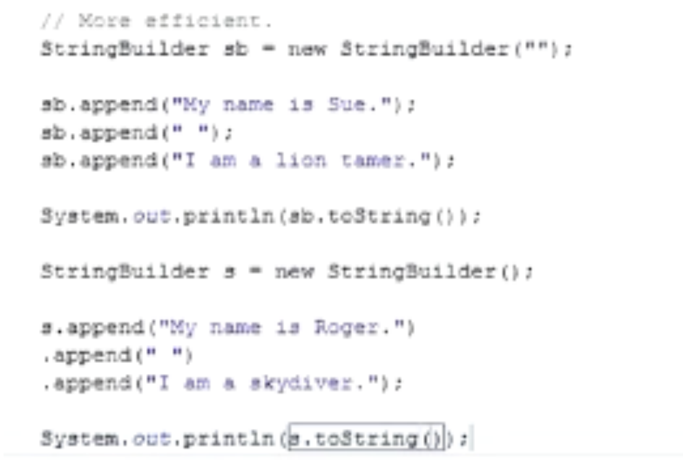
Commercial
- SpiritCache (from SpiritSoft)
- Coherence (Tangosol)
- ObjectCache (ObjectStore)
- Object Caching Service for Java (Oracle)

In Java , strings are immutable.. in example below, string info is changing which is not a good practice



```
// Inefficient
String info = "";

info += "My name is Bob.";
info += " ";
info += "I am a builder.";

System.out.println(info);

// More efficient.
StringBuilder sb = new StringBuilder("");

sb.append("My name is Sue.");
sb.append(" ");
sb.append("I am a lion tamer.");

System.out.println(sb.toString());
```

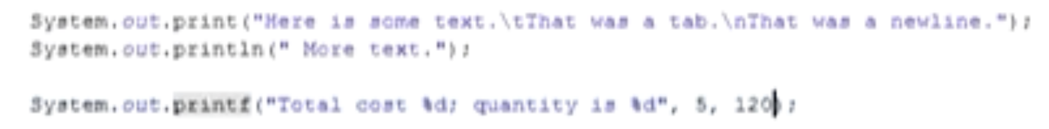
Another method is as shown below



```
// More efficient.
StringBuilder sb = new StringBuilder("");

sb.append("My name is Sue.");
sb.append(" ");
sb.append("I am a lion tamer.");

System.out.println(sb.toString());

StringBuilder s = new StringBuilder();

s.append("My name is Roger.")
.append(" ")
.append("I am a skydiver.");

System.out.println(s.toString());
```

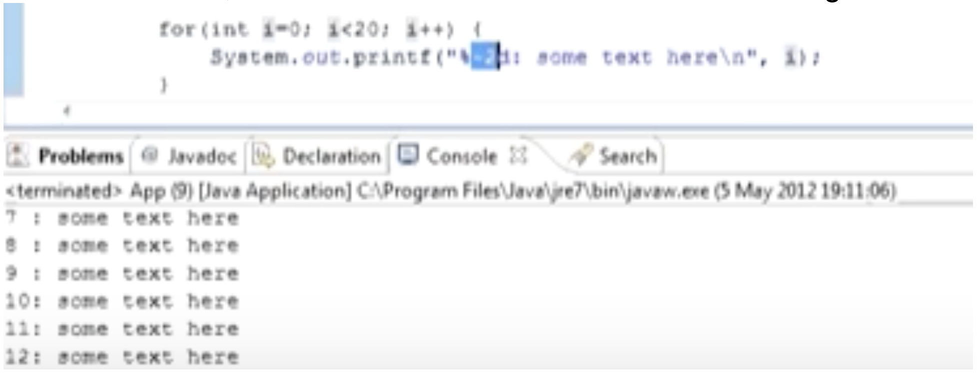
So basically there was older class called Stringbuffer.. new one is StringBuilder as shown above. But both are same.

Difference between system.out.print and system.out.println is that println will print on next line ..where printf is for formatting .. as shown below. So basically printf expects us to specify formatting character and arguments in the bracket



```
System.out.print("Here is some text.\tThat was a tab.\nThat was a newline.");
System.out.println(" More text.");

System.out.printf("Total cost %d; quantity is %d", 5, 120);
```

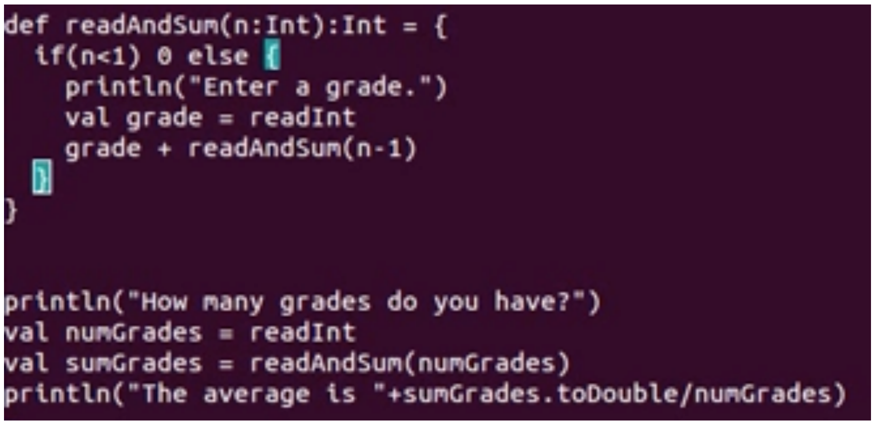
Additionally. it is also possible to specify extra flags between % and d sign to format text as shown below, default size of number is 2 and - for left aligned



```
        for(int i=0; i<20; i++) {
            System.out.printf("%-2d: some text here\n", i);
        }
```

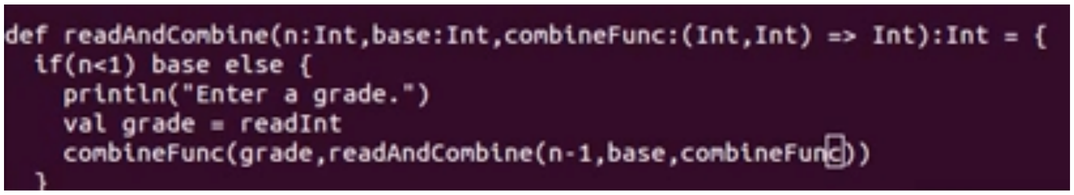
```
<terminated> App (9) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 May 2012 19:11:06)
7 : some text here
8 : some text here
9 : some text here
10: some text here
11: some text here
12: some text here
```

———————Scala —Continuation from Information notes 2.
What is recursive function. As shown below, if same function is called within same function body then it is recursive function

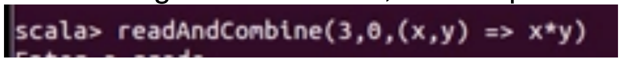


```
def readAndSum(n:Int):Int = {
  if(n<1) 0 else {
    println("Enter a grade.")
    val grade = readInt
    grade + readAndSum(n-1)
  }
}


println("How many grades do you have?")
val numGrades = readInt
val sumGrades = readAndSum(numGrades)
println("The average is "+sumGrades.toDouble/numGrades)
```

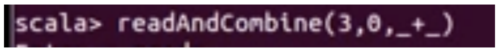
Recursive function can be passed into argument as shown below , as well in addition to passing to body of the function .



```
def readAndCombine(n:Int,base:Int,combineFunc:(Int,Int) => Int):Int = {
  if(n<1) base else {
    println("Enter a grade.")
    val grade = readInt
    combineFunc(grade,readAndCombine(n-1,base,combineFunc))
  }
```

and while executing above function, we can pass function literal definition itself as shown 

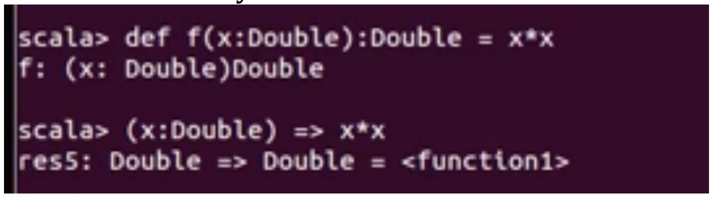
```
scala> readAndCombine(3,0,(x,y) => x*y)
```

or 

```
scala> readAndCombine(3,0,_+_)
```

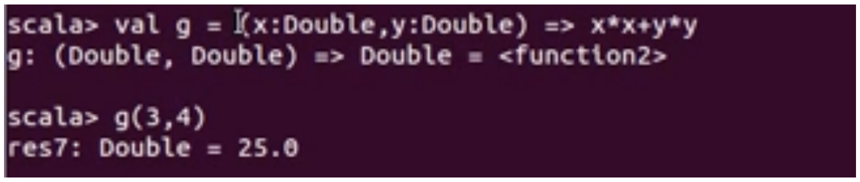
So in short, while execution, we are passing what function inside body should do , either sum or multiplication. So no need to define separate sum or multiplication inside body of the main function.

There are two ways to use functions as literals . 1. Using DEF and 2. is as shown below



```
scala> def f(x:Double):Double = x*x
f: (x: Double)Double

scala> (x:Double) => x*x
res5: Double => Double = <function1>
```

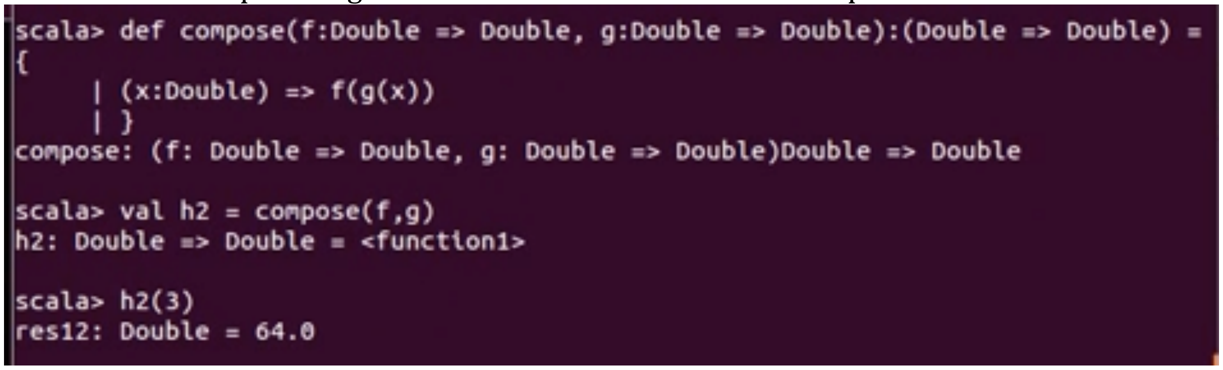
So here (x:Double) => x*x is called function literal and it has type Double Double. Input and output types.



```
scala> val g = (x:Double,y:Double) => x*x+y*y
g: (Double, Double) => Double = <function2>

scala> g(3,4)
res7: Double = 25.0
```

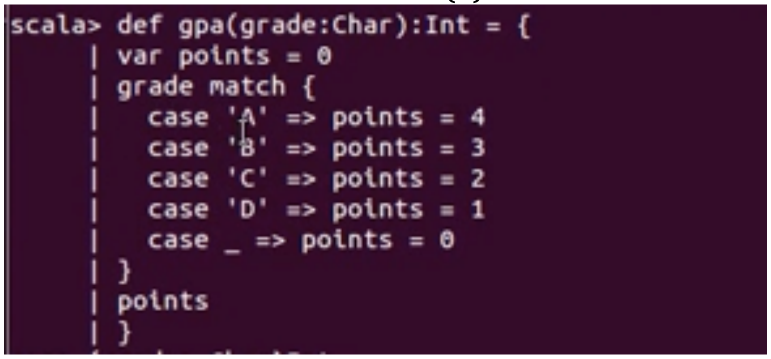
Basically these function literals used in higher order functions. Higher order function takes another function as argument and return output as function.

Below is the example of higher order function and our own compose.



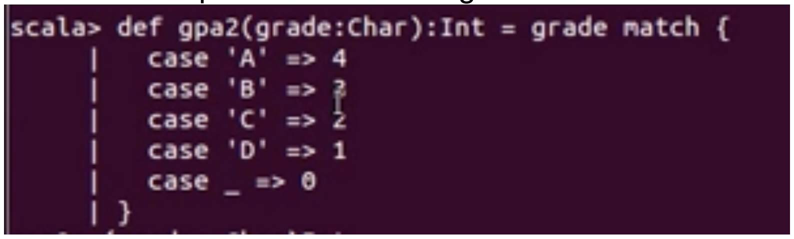
```
scala> def compose(f:Double => Double, g:Double => Double):(Double => Double) =
{
     | (x:Double) => f(g(x))
     | }
compose: (f: Double => Double, g: Double => Double)Double => Double

scala> val h2 = compose(f,g)
h2: Double => Double = <function1>

scala> h2(3)
res12: Double = 64.0
```

Match as a statement below so { } is block of statement



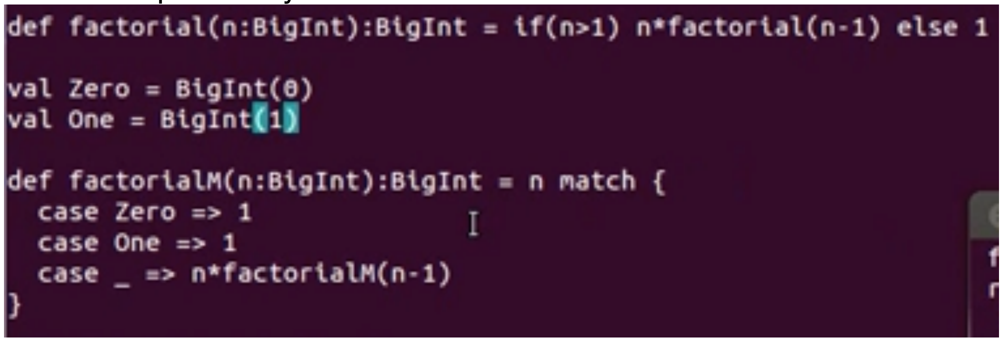
```
scala> def gpa(grade:Char):Int = {
     | var points = 0
     | grade match {
     |   case 'A' => points = 4
     |   case 'B' => points = 3
     |   case 'C' => points = 2
     |   case 'D' => points = 1
     |   case _ => points = 0
     | }
     | points
     | }
```

Match as a expression below so grade match defines a expression



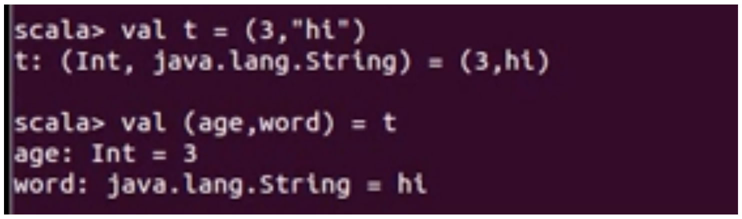
```
scala> def gpa2(grade:Char):Int = grade match {
     |   case 'A' => 4
     |   case 'B' => 3
     |   case 'C' => 2
     |   case 'D' => 1
     |   case _ => 0
     | }
```

here is couple of ways we can code factorial



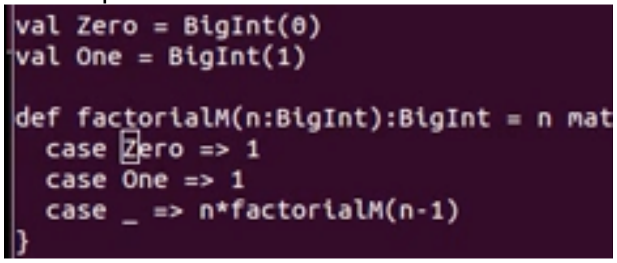
```
def factorial(n:BigInt):BigInt = if(n>1) n*factorial(n-1) else 1

val Zero = BigInt(0)
val One = BigInt(1)

def factorialM(n:BigInt):BigInt = n match {
  case Zero => 1
  case One => 1
  case _ => n*factorialM(n-1)
}
```

What is Patterns -
As shown below , lower case words are always use for variables . so age and word is variables.



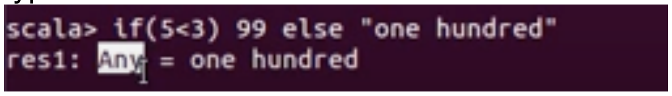
```
scala> val t = (3,"hi")
t: (Int, java.lang.String) = (3,hi)

scala> val (age,word) = t
age: Int = 3
word: java.lang.String = hi
```

If we use capital letter like Age , instead of age, then in scala it is assumed that it is value defined somewhere else . here is same example where Zero and One is defined with capital letter



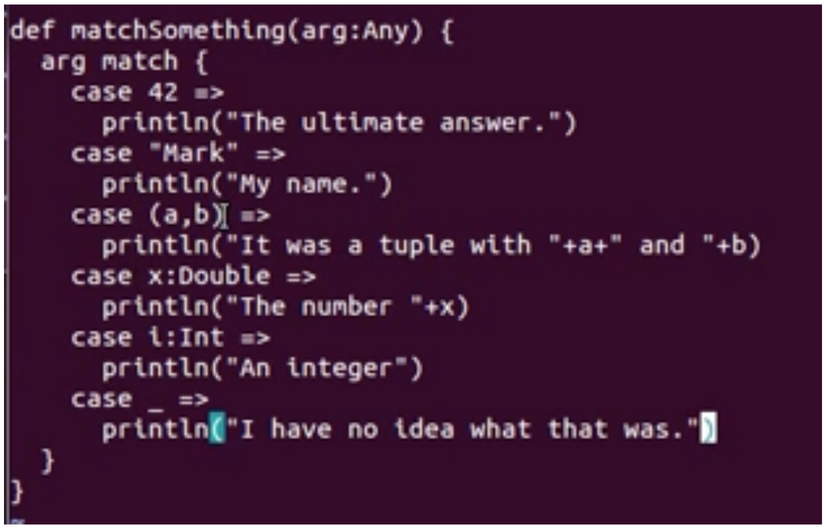
```
val Zero = BigInt(0)
val One = BigInt(1)

def factorialM(n:BigInt):BigInt = n mat
  case Zero => 1
  case One => 1
  case _ => n*factorialM(n-1)
}
```

Introduction of type ANY .. in cases where scala cannot figure out the output type then it gives ANY as shown below.. If condition has two values depending if it is true or false , 99 is integer and “one hundred” is string .. since it is not consistent, scala returns ANY type



```
scala> if(5<3) 99 else "one hundred"
res1: Any = one hundred
```

What are patterns — using ANY . As shown below, arg accept any input from user and then treat it any data type , tuple, or string , int what ever is under case statement.



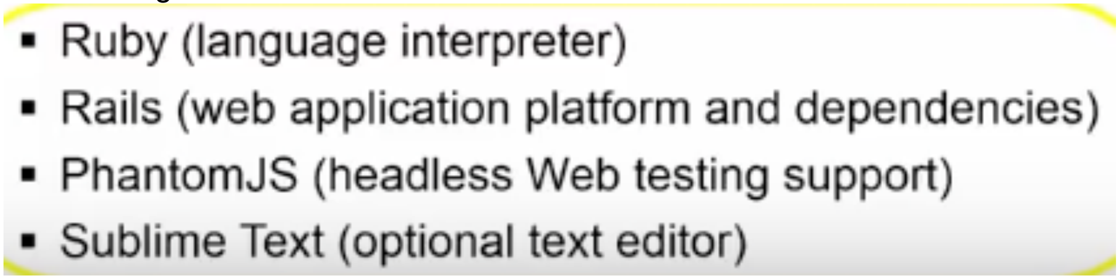
```
def matchSomething(arg:Any) {
  arg match {
    case 42 =>
      println("The ultimate answer.")
    case "Mark" =>
      println("My name.")
    case (a,b) =>
      println("It was a tuple with "+a+" and "+b)
    case x:Double =>
      println("The number "+x)
    case i:Int =>
      println("An integer")
    case _ =>
      println("I have no idea what that was.")
  }
}
```

————How to code TRY and CATCH

——————What is Ruby on Rails -

RoR is open source web framework that is written in Ruby programming language that allows to build powerful web applications. It consists of
1. MVC Architecture (Model view controller) - this separate application into 3 distinct parts
	a. Model - Ruby classes that handles business logic in your application
	b. View - This renders data from model and handles presentation of data
	c. Controller — This sits between model and view . This controls the flow of the application . It handles request and make changes to model etc
2. Convention over configuration - These are like set of defaults
3. DRY (Don't repeat yourself)

So following tools are used -



- Ruby (language interpreter)
- Rails (web application platform and dependencies)
- PhantomJS (headless Web testing support)
- Sublime Text (optional text editor)

There are two managers as mentioned below
1. RVM - Ruby version manager - older version software
2. RBENV (with Ruby build plugin)

we use gem (gem is package manager just like RPM or APT package manager) command instead of yum to install ruby , rails
Check digital ocean link for more details on installation