Overwhelming Java Review

- ✓ The order of size of the variable numerical types

  Byte
  Short
  Int
  Long

- ✓ We can use Random class to create objects to generate random variables int or any other type
- ✓ Enhanced for loop

```
int ints [] = {2, 3, 5, 7, 8};
for (int i: ints) {
    System.out.println(i);
}
```

  We use i directly as the element of the array

- ✓ we use manipulate \t to put a tap in the text of the output
- ✓ how to copy the value only of an object not the reference

```
ArrayList <Integer> y = (ArrayList) x.clone ();
```

- ✓ when you don't know how many the arguments of the method

```
public static int change (int...numbers)
    {
    int total = 0;
    for (int i: numbers) {
    total += i;
        }
    return total/numbers.length;
    }
```

- ✓ String.format("arguments in printf style", arguments)

  EX;

```
String.format ("%d good %d now %d display2 ", h, m, s);
```

  returns string with wanted format

- ✓ we can redefine the defined methods for our class objects like `toString()` or `equal()` but you should have the same return type
- ✓ if you need to use an object as an string in any code

  You should provide overloaded toString () function to make it clear what to do with this object

- ✓ we can make any class inherit anther class by using keyword `extends` after class name
- ✓ we have two super class types:

1- Direct super class
2- Indirect super class

- ✓ there is a default class in java called Object class when you make any class you class extends the object class without seeing so, so Object class is the highest super class
- ✓ if a class B extends class A so, if B doesn't contain any constructor the complier will assume it has:

```
B ()
{Super ();
}
```

  But if A doesn't have empty argument constructor and has anther constructor with arguments so B should has constructor takes like this arguments

- ✓ any package in java can contain other packages or java classes
- ✓ Access modifiers:
  1- `Private` cannot use the data member of function outside the class even in sub classes
  2- `Protected` can only use this data member or function inside the class or any subclass of this class or and class in the package of the class containing this member
  3- `Public` can be accessed from any place the project
  4- `Default` if we didn't specify any access modifier the data member of method can only be access form the classes inside the package containing this member
- ✓ it's better to set data with private of protected access modifier and use set and get function to use data members
- ✓ static make the data member or the method is common between all objects of the class and can be called with any object or by the name of the class even we didn't create any object of the class
- ✓ we cannot call non static variable of our class in a static method in our class so we can't use this in the static method
- ✓ static members are called class members
- ✓ notice that the main function is static so we can call it by the code using:
  `yourclassname.main (args);`

- ✓ methods inside a class can have the same name but must have different arguments numbers or types this called method overloading
- ✓ we can rewrite a super class method in a sub class and it's called overriding
- ✓ `final`
  1- data members values cannot be changed but can be initialized by anther variable's value and we can but the value of a final data member in the constructor of the class
  2- methods cannot be overridden
  3- class cannot be extended or inherited
- ✓ if i want to get all the classes in a package we use *
  `import java.util.*;`
- ✓ using scanner class

```
Scanner x = new Scanner (System.in);
 int good = x.nextInt ();
```

- ✓ composition is when you define a user defined object as a data member inside another class
  EX:

```
class x {
int n1;
int n2;
myclass n3 ;}
```

- ✓ in try catch → in catch we define Exception object which will contain data about the error has taken place
- ✓ Exception is a general class all Exception classes are inherited for it
- ✓ to know the type of the Exception
  `System.out.println (now);`
  Or
  `System.out.println (now.getMessage ());`

- ✓ catch block must work if the Exception of type of its argument

  EX:

  ```
  catch (InputMissMatchExcepion e) //also ArithmaticException
  ```

  Won't work until an input mismatch exception takes place
- ✓ we can write an empty catch to get rid of the exception without taking an additional steps
- ✓ finally caption of code is done even if we had exception of not

  ```
  try
  { }
  catch (Exception E)
  { }
  finally
  { }
  ```
- ✓ finally caption of code will be hold even if catch contains `return;` statement
- ✓ to make our method throws an Exception at our defined state

  ```
  throw new Exception ("your Exception message");
  ```
- ✓ in java we have two Exception thrower types

  1- Exception catcher

  the method will throw the Exception and will solve the problem

  The throw line must come in try caption and then a catch caption

  EX:

  ```
  try {
  throw new Exception ("your Exception message");
  }
  catch (Exception e)
  {
  e.getMessage ();
  }
  ```

  2- Exception propagator

  The method throws the Exception and won't solve the problem

  We must mention throws Exception in definition of the method

  ```
  public int show (int i) throws Exception
  {
  if (i > 101)
  throw new Exception ("your Exception message");
  }
  ```
- ✓ we can specify more than one Exception type in the throws statement

  EX:

  ```
  public int show (int i) throws InputMissMatchExcepion,
  ArithmaticException
  ```
- ✓ we can use multi catch and it can be with different exception types

  So we can use InputMissMatchExcepion or ArithmaticException in the place of Exception
- ✓ there is a default Exception type called runtime Exception so you maynot need mention it in thorws

  ```
  public int show (int i)
  {throw new InputMissMatchExcepion ("Your message") ;}
  ```

  Notice that InputMissMatchExcepion is one of Runtime Exception

- ✓ Exceptions have two types
  1- Checked
  It's detected while compiling the program before running it
  2- Unchecked
  It's runtime one so it's detected while running the program
- ✓ you can create your own Exception class
  1- Create a normal class which `extends Exception` class
  2- Add public method with `super ("your message");`

```
public myExceptionMes (String mes)
{
super (mes);
}
```

  You can modify this class as a regular class
- ✓ `assert` is a way in java to detect logical errors
  To make it available go to
  Run --> customize --> VM option and write `-ea` to make it available
- ✓ `assert` check a condition you give it, if the check is not true it throws an Exception called AssertException
- ✓ the best use of assert is when we use it for a very long logical expression to check its result
- ✓ to write assert in code

```
assert xx> i;
```

- ✓ if we have more than one assert we can make it unique by a message we define to be sent in the error

```
assert x>y: "your Error message";
```

- ✓ polymorphism:
  When a method of super class has overridden methods so the same call in code causes different method
  EX: class b extends a and c extends a

```
ourmethod (a valuea)
```

  Can also take

```
ourmethod (b valueb)
```

  And

```
Ourmethod (c valuec)
```

- ✓ we can assign a new subclass object to a super class reference

```
mysuperclass x = new mysubclass ();
```

  The x object will only have the data members and methods of the super class but it will take the overridden methods from the sub class even if the overridden methods in sub class will use other data members of methods form the subclass but we cannot assign a new superclass object to a sub class reference
- ✓ Recursive methods are good to use in complex operation like tree or complex data

- ✓ three way to name variable
  1- C style: all letters small and using underscore
  EX: set_name
  2- Pascal style: all words starts with capital letter
  EX: SetName
  3- Cannal style: middle words starts with capital letter
  EX: setName
- ✓ it's important to put full comments in your class to make it easy to fix and use it
- ✓ abstract class:
  We cannot make new object of this class it can only be used as a super class of other class and it's important to use it in inheritance or in polymorphism
- ✓ Any class that is not abstract is called concrete
- ✓ notice that abstract class can have static data members and method and we can use it by class name without creating any new project
- ✓ abstract methods can only be written inside in abstract class and it's a method without body inside abstract class and sub classes will write its body
  ```
  public abstract void showing ();
  ```
- ✓ final and static methods cannot be abstract methods
- ✓ constructors cannot be abstract
- ✓ we can use abstract class as a type of argument or return for a method
- ✓ instanceOf is way to know the type on an object in the code to take some decision due to the result
  ```
  if (objectx instanceOf Classname)
  ```
- ✓ you should know that any object is an instance of its sub class and also from its super class so instanceOf operator will see it as an instance of its all super classes and its class
- ✓ interface
  1- Create a normal class and replace class keyword with interface
  ```
  public interface Myinterface {
      int show ();
      void good ();

  }
  ```
  2- Interface contains some methods without body like abstract methods put without abstract keyword
  3- Create your class to use and use implements
  ```
  public class third extends first implements Myinterface {}
  ```
- ✓ any class implements an interface must write body for all of its methods
- ✓ any class can inherit anther class and implement an interface in the same time
- ✓ we can use interface as an argument or return type for a method and we can use any object of class implements this interface
- ✓ we cannot create new object of an interface so we cannot use new keyword with it
- ✓ we can make any class implements more than one interface
  ```
  public class myClass implements Myinterface, Myinterface2 {}
  ```
  And then any object of this class can be used instead of any of the two interfaces

- ✓ any variable will be defined inside an interface must have a value and will be treated as a final static variable and it can be used by the name of the interface or any object of it or by any class implements the interface
- ✓ we use an interface reference to hold an object of a class implements this interface and with this reference we can access any interface member using the implementation of the class
- ✓ final and abstract classes can implement interfaces
- ✓ if a super class implements an interface so all its sub class also implement this interface using the overridden methods in the super class and we can override it again in the sub classes
- ✓ we can make an interface inherit anther interface

```
public interface Myinterface2 extends Myinterface
```

- ✓ java doesn't allow multiple inheritance but allow multiple interfaces
- ✓ anonymous class with interface we can create class without name and make it implement an interface we want

EX:

```
Myinterface i = new Myinterface () {
     @Override
     public int show() {
     System.out.println ("i am nonamous");
     return 10;
     }
     @Override
     public void good() {
          System.out.println ("good nonmous");
     }  };
mytest (i);
```

Notice we have used i as an argument for the method mytest () which takes Myinterface object

- ✓ notice we can write it in another way so we can use it as a direct argument for a method

```
     mytest (new Myinterface () {
     @Override
     public int show() {
     System.out.println ("i am nonamous");
     return 10;       }
     @Override
     public void good() {
     System.out.println ("good nonmous");}});
```

In this step we have made object without name from the interface and we have made a class to implement without name

- ✓ also we can call a method in an interface we will call it without name and make class implement it and create object of this class without name

```
new Myinterface () {
    @Override
    public int show () {
    System.out.println ("i am nonamous");
    return 10;      }
    @Override
    public void good () {
    System.out.println ("good nonmous");
    }}.show ();
```

- ✓ we can make the above three steps for an abstract class and with its abstract methods with the same steps and syntax
- ✓ we can cast primitive data types and also user defined data type
  Also we can cast an interface to an object of a class which implements this interface
  EX1:

```
subcalss t = (subclass) new superclass ();
```

  Or

```
subcalss t = (subclass) superclassObject;
```

  Ex2:

```
yourclass t = (yourclass) youinterfaceobject;
```

  So we can use any of the data members of the sub class or the class which implements the interface
  We can do this between Object class –the highest super class - and any other class
- ✓ notice if we cast an object of class which implements an interface to anther object of anther class that implements the same interface we will get error

```
Myinterface x = new Myclass1 ();
Myclass2 y = (Myclass2) x; // this will give you an Error
```

- ✓ we can create an Empty interface to make a method accept all class that implement the Empty interface as an arguments or return types
- ✓ android example of using inline interface, class definition and object without name

```
next.setOnClickListener (new OnClickListener () {
            @Override
            public void onClick (View v) {
            Intent i = new Intent ();
            i.setAction ("android.intent.action.center");
            startActivity (i); }});
```

- ✓ enumeration:
  It's a way to collect group of constants in one place
- ✓ to make new enumeration go to package press new and choose java enum
- ✓ write your constant name inside the enum and , between them and in they are regarded as final static variables the end write ;
  EX:

```
public enum Weekdays {
    sunday, monday,saturday,tuesday,wednesday,thursday,friday;}
```

- ✓ we can also define it inside any class and there it can take any access modifier we want like private or protected

  Note: enum inside class works as it's static so it can be used by class name and it's common between all objects
- ✓ to put parameters for enum constants

  1- Define a constructor for the enum and it must be private as we cannot create new object of the enum

  2- The constructor must take argument of the type of the data you send to its constants and it must be private as we won't create any new element of this Enumeration

  3- All constants of an enum must take parameters of the same type

  4- We also should define a variable inside the enum to use it in constructor and it will be of the same type of the constant parameters

  5- If you will pass parameters to enum elements you must set it for all enum elements

  EX:

```
enum Weekdays {
sunday(3,"work"),
monday(3,"fast"),saturday(3,"start"),tuesday(3,"play"),wednesday(3,"p
ray"),thursday(3,"fast"),friday(3,"rest");
public int val;
public String xtra;
private Weekdays (int v, String x)
{this.val = v;
this.xtra = x ;}
}
```

- ✓ to use enum constant enumName.enumconstantName

```
System.out.println (Weekdays.friday);
```

- ✓ notice we can define public data members and methods inside enum and we can call it by the constant name of the enum and each constant of enum has its copy of this data members and methods

  1- Here we get a data member

```
   System.out.println (Weekdays.friday.xtra);
```

  2- Here we get a method

```
 Weekdays.friday.show ();
```

- ✓ very smart note:

  As all enum constants are static final so we can access one of them form any other constant

  EX:

```
weekdays.friday.saturday.xtra;
```

  Notice with this statement we get the value of xtra variable for Saturday constant not Friday one
- ✓ we can send a user defined class object or an interface as a parameter for enum constants but we must make a variable of it in the body and use it in the constructor

  EX:

```
enum Weekdays {
Sunday (new myClass ()), Monday (new myClass ());
 public myclass foreign;
 private Weekdays (myClass f)
 { this.foreign= f ;}}
```

- ✓ notice: we cannot make an enumeration inherit anther one but we can make an enumeration implement an interface

```
public enum monthdays implements Myinterface {
enumConstant1, enumconstant2;
@Override
public int myInterfaceMehtod () {     } }
```

And so we can send any of this enum constant to a method that takes interface which the enum implements and also we can make it as a return type

- ✓ we can make abstract method inside the enum but we must override this method for all constants inside the enum

EX:

```
public enum monthdays {

     help
     {
     @Override
     public void milk ()
     {System.out.println ("help") ;}
     }
     ,
     work
     {@Override
     public void milk ()
     {System.out.println ("work") ;}
     }
    ;
public abstract void milk () ;}
```

Green: constant names

Yellow: overriding the method milk for each constant

Blue: comma between elements until end use ";"

Pink: milk declaration as an abstract method

Now each constant has its copy of the milk method

- ✓ to define an object of the enum we use this syntax

```
myEnum x = myEnum.enumConstant;
```

EX: `monthdays x = monthdays.work;`

- ✓ we can use enum in switch

```
switch (x)
   {
   case help:
   System.out.println ("enum switch help");
   break;
   case work:
   System.out.println ("enum switch work");
   break ;}
```

X is an enumeration

And "help" and "work" are the constants inside this enumeration

- ✓ java compiler make a method for each enum called values()

This values method returns array contains all constants of the enum with the order you have written it in the enum

```
myEnum [] x = myEnum.values ();
```

EX:

```
monthdays [] xyz=monthdays.values ();
```

- ✓ to make enhanced for for the enum

```
for (monthdays x: monthdays.values ()) {
x.milk ();    }
```

First define object of type of the enum and but the values () method after ":"

- ✓ java compiler make a method for each enum called valueOf()

This valueOf method takes the name of the constant and returns the constant as an object

This way is good to get the constant by name and use its data members and methods

When we give it wrong name it throws exception called IllegalArgumentException we should call it in side try...catch caption

```
Weekdays.valueOf ("sunday");
```

- ✓ notice we cannot add new constant for enum or adjust its constant names in our use of it but we can adjust enum constant variables values

```
Weekdays.sunday.xtra ="advance";
```

- ✓ if a method takes or returns an object of Object class so it can take any object of any class as Object is the highest super class of all classes of java
- ✓ in java there is a class called class
- ✓ for any object of any class there is a method called getclass()

This method returns object of class class and we can use this method to get alot of data about the class of the object like its name or methods

EX:

```
first f = new first ();
System.out.println (f.getClass ().getName ());
```

- ✓ if a method returns an object of a class we can put dot after method call and we can use it as the return object

EX:

Method build

```
public static first work ()
{     return new first ();    }
```

Method call

```
work ().show ();
```

- ✓ also we can use this method as a reference to save data

EX:

Method build

```
public static first work (first x )
    {     return x;     }
```

Method call

```
first f = new first ();
work (f).x = 10;
System.out.println (f.x);
```

- ✓ nested classes:

Java allows you to build class inside anther class and you can add data members and methods to the inner class

- ✓ we can add any access modifiers we want to the inner classes like private, public or protected
- ✓ inner class is a normal class can contain constructor, data members and methods with different access modifiers

- ✓ nested classes types:

  1- static

  >> Write static before the class name

  >> Static class can only use static data members and static methods of the outer class

  >> We can create static and nonstatic data members and methods inside the static inner class

  >> To define object of an inner static class in code

```
outerClassName.innerClassName x = new outerClassName.innerClassName
();
```

  Ex:

```
first.car x = new first.car ();
```

  2- Nonstatic (called inner)

  >> Inner class can use any data member of the outer class static or non-static

  >> We cannot create static methods inside the nonstatic inner class but we can create static data members

  >> You must create an object of outer class to create an object of inner class

  >> To define object of an inner nonstatic class

```
first x = new first ( );
first.car  y= x.new car ();
```

  Notice must use first object to get new car object

- ✓ local inner class:

  In java you can define a class inside a method; we cannot use access modifiers with the local inner class

- ✓ local inner class can normally inherit any class and implement any interface
- ✓ anonymous inner class:

  We can make inner local class without name and make it extend anther class

  Syntax:

```
 new yourTargetClassName ()
{
Any methods or data members you want to add to this anonymous class
};
```

  EX:

```
public static void work ( )
{
new first () {
int i = 10;
void working ()
{  }
};}
```

Yellow: the class extended to our anonymous class

Pink: new data members and methods for our anonymous class

Blue: method we define anonymous class inside

- ✓ also you can access the data members and methods of the extended class in the same line

  Syntax:

  new yourTargetClassName ()

  {

  Any methods or data members you want to add to this anonymous class

  }.youTargetClassMethod ();

  EX:

```
public static void work ( )
{     new first () {      }.dummy ();   }
```