Shortcut key in Eclipse:

1. sysout(ctrl+spacebar) system.out.println();
2. object class is root(head) class in java classes
   **So every class of Java is the child class of the Object class.

3.abs(int) for integer type, labs(long) for long type , fabs(float) for float type.

4. float takes 6 decimal places and double takes 15 decimal places.

5.In command prompt we can run a JAVA program in any specific version.
   i.e javac -source 1.4 class_name.java

.jar FILE:

1.export the project and choose a jar runnable file ,then select project_name and select path location.

2.After creating a jar file we can run the jar file in the command prompt.
   cd desktop (any file_path)
   ls *.jar (for checking .jar file is available or not)
   java -jar jarfile_name.jar (for execute the jar file)

DataBase Connection:

(1)In command prompt for connect with DB we have to write:
   (i) sqlplus
   (ii)user_name (system)
   (iii)"password" ("Nayak@123") //If special symbols then "inside String"

byte=8 bits
short=16 bits
integer=32 bits (range--> -2^31 to 2^31-1)
long=64 bits

float=32 bits
double=64 bits with decimal value

Wrapping of Data:

int --->float => int num1=5, float num2=num1;
int --->double=> int num1=5, double num2=num1;
int --->String=> int num1=5, String str1=Integer.toString(num1);

String --->int=> int str1=55, int num1=Integer.parseInt(str1);
double --->int=> double num1=12.234 , int num2=(int)(num1)

float --->int=> float num1=12.23 , int num2=(int)(num1)

This:

This is the reference variable that refers to the current class object.

Super:

Super is the reference variable that refers to the parent class object.

Final:

- final mused in variable,method & class .
- final variable value not be changed, final method can't override,
- final class can't inherit (can't extend).
- Not initialize the final variable ,we can only initialize the value in Constructor.
- If we declare any parameter as final, we cannot change the value of it.

Instanceof:

It's given the boolean value(T & F), If instance is of the Class.

Interface:

It is used to achieve abstraction and multiple inheritance in Java.
And also it gives security.

Wrapper Class:

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

boolean	Boolean
char	Character
Byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Syntax:- int num=20;
Integer x=Integer.valueof(num); //explicitly
Integer y=num; //autoboxing by compiler internally

Syntax:- Integer num=20;
int x=num.intValue();

Durga Soft Solution:

1.A java program can have a lot of classes but public class at most 1.(either 1 public class else no public class).
File name always as the used class name which is public, else we can give file name of any class name or anything u want.

2. If a class is not having a main method,if we execute the .class file then error should come ie. The main method is not found.

3.import Statement(Packages):

package:
It's a group of similar types of class,interface and sub-packages.
In command prompt we can run package as below command:-
cd desktop
javac -d . filename.java //(First we have to compile all classes(.java) file using this command)
java package name.filename
We can import a package in another package like below syntax:
import package name.*;
OR import packagename.classname(previous package classname);
OR pack.A obj = new pack.A(); //using fully qualified name
means packagename.classname object=new packagename.classname();

Explicit import:- import java.util.ArrayList; (If u use single name class like only ArrayList)
Implicit import:- import java.util.*; (If u use Multiple Number of classes like ArrayList,Iterator,Collection...etc)
a. import java.lang : we need not to import java.lang package ,by default it can access from java.lang package.
(String,StringBuffer/Builder,Exception,Thread...etc).

b. default package : If we want to access data from another package then we have to import the package name.

(import package_name.*;)

But if the file location is the same and different classes are in one package(But not using any sub_packages like util,io,regex...etc),then there is no need to import anything. We can access methods of different classes by creating an object of that class.

If you want access sub_packages like util,io,regex...etc then we have to import.

import java.*; means By it's we can access only classes ,interfaces....etc But not sub_package.

import java(package).util(sub_package).regex(sub_pachage).pattern(sub_package) is used for accessing subpackages.

We can use only one package inside a source file(.java file). (ex:- package DNayak).

4.Access Modifier:

For top level classes there are 5 types of access modifier like:-
(public, abstract,default, final,strictfp)
For inner classes there are 5+3=8 types of access modifier like:-
(public, abstract,default, final,strictfp, private, protected, static)

**Always 1st check the class level modifier then go to member level modifier

Example:-
```
public class A{//top class
  private class B{//inner class
  }
}
```

Access Modifier	within class	within package	outside package by subclass/child only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

** A class cannot be private or protected except a nested class.

(Hiding the internal implementation and just highlight the set up services) + (security reason)

5.Abstract Method and Class: (But variable is not abstract)

(i) The method which are declared but not defined are called abstract method, and class will be abstract if
an abstract method is present in that class.

(ii)If a class does not have any abstract method and class is not defined properly,then also it is called abstract class.

Example:-(.t.i.e Domi/empty definition)

```
abstract class Test{
    public void m1()
     {    }    //method m1() and m2() have not any definition,so no need to create object for class to calling them. So here class is abstract.
    public void m2()
     {    }
   }
```

(iii)Partially defined classes are called Abstract Classes.

(iv) Objects can not be instantiated in abstract class.

(v)If Parent class is abstract and having abstract method, then definition of abstract method will be done in Child class.All abstract method definition should to present in child class, If not available then we have to define in next child class.

Example:-

```
abstract class Test{
    public abstract void m1();
    public abstract void m2();
  }
 abstract class SubTest extends Test{
    public void m1() {}
    //public void m2() {}  //We have to define all abstract method in child class
  }                    If you not define then you have to define in another child class.
 abstract class SubSubTest extends Test{
    public void m2() {}
 }
public class MyRough {

    public static void main(String[] args) {
  }
  }
```

**(vi)Child class method may be defined or may not be.But if we use Abstract method in parent abstract class, then we must have to define the method which are present in child class.

6. Public Modifier:

(i) For access from anywhere we use a public modifier with class,method,variable.....etc.

**(ii)If the method is public in a package but class is not public, Then we can't access the method in

in another package.

7.Protected Modifier:

(i)We can access within the class and package by creating parent class or child class references objects.

i.e A obj=new A(); or A obj=new B(); or B obj=new B(); .(Here A=parent , B=Child).

(ii)We can also access protected methods/variables outside the package but in child class(extends) only.

Here we can only create references of child class But not parent class.

i.e B obj=new B();

8. Interface:

(i) Interface is the Blue print of Class.It has static constants and abstract methods.

(ii) It is used to achieve abstraction and multiple inheritance in Java.

(iii)There can be only abstract methods in the Java interface, not method bodies.

**(iv) Whenever we implement any interface method ,then it is compulsory we should declare the method as public.

(interface methods are by default public or abstract).

(v) If a interface having two or more abstract method, then we have to define all the method inside

the class. (class which implements the interface)

(vi)If we not want to define any method of the interface then we have to mention the class is abstract

(abstract class means partially defined).

9. Data Hiding: (It's only for security)

(i) For data hiding method must be a private modifier.

(ii) And giving username and password in public modifier.

i.e

```
class Account{
    private double balance;
}
class BalanceDeatils{
        public double getbalance() {
      //Here username  and password.
    //After validation balance will display
        return balance;
}
  }
```

**(iii) Security (Secure)
Enhancement (We can change any thing in our backend code without problem to the user (i.e ATM))
Maintainability and Modularity.

(Data Hiding +Abstraction) + {getmethod() + setmethod()}

10.Encapsulation:

(i)Grouping of data members(variables) and corresponding methods in a single unit(inside a class)
is called Encapsulation.
(ii)Encapsulation = Data Hiding +Abstraction
(In ATM balance is private i.e Data Hiding, And get and set methods are internally defined i.e abstraction)
(iii)Outside people only access the getter and setter method only, But not private members/variables.
(iv)Advantages: "Security" Disadvantage: LOC bcz of get & set method.
**(v)If every "variable" member is private in a class, then it's called a tightly Encapsulated Class.
If the parent class is not private,then the child is not a tightly encapsulated class.

Example1:-
```
class A{
    int a; //not tightly encapsulated
}
class B extends A{
    private int b;//not tightly encapsulated because it inherited the parent class variable(i.e int)
}
class C extends B{
    private int c;//not tightly encapsulated because it inherited the parent class variable(i.e private & int)
}
```

Example2:-
```
class A{
    private int a;
}
class B extends A{
    private int b;          //Here all classes are tightly encapsulated bcz all have private variables.
}
class C extends B{
    private int c;
}
```

11.Inheritance:

(i)It's called IS-A Relationship.

(ii)Code Reusability.

(iii)"extends" keyword is used for implements of class.

(iv)Multiple inheritance and Hybrid inheritance is not supported in java through class. (only support by interface)

(Because of ambiguity or diamond access problem).

**(v)Multiple Inheritance:

It's supported only through interface.If there are method names that are the same,then we can call a method But definition of method is once in child class.

Example:

```
interface A{
        void m1();
  }
interface B {
  void m1();
    }
  class C implements A,B{
        public void m1() {
              System.out.println("World");
              }
        }
  class MyRough {
        public static void main(String[] args) {
              B obj=new C();
    //A obj=new C();
    // C obj=new C();
              obj.m1();
        }
  }
```

(vi)But in python multiple inheritance is supported,by the sequence of parent class.(first parent class will be called).

Example:

```
class P1:
    def m1():
        print('p1 class method is called')
class P2:
    def m1():
        print('p2 class method is called')

class C(P1,P2): //P1 class called,Here depend upon sequence of parent class.
```

```
                    //If class C(P2,P1) ,then P2 is called.
                pass

            obj=C()
            obj.m1()
```

(vii)cyclic inheritance does not work in all languages, It gives a compile time error.

i.e class A extends A or/ class B extends A
class A extends B

12.Method Signature:

(i)Method signature means method "name" and it's argument type. (data type,order of data type,no of arguments).

i.e public int m1(int i,float f); //Here method signature is m1(int,float)

13.Method Overloading:(compile time polymorphism):

(i) Method name must be "same" and type of arguments,orders of arguments,no of arguments must be "different".

**(ii)Automating Promotion: (if method argument type is not available,then also compiler check for next level.)

ie. It's shown in my Overloading Example in Eclipse IDE.

automating promotion chart:(always check next level data type for auto promotion)

byte ->short ->int ->long ->float ->double.
char ->int ->long ->float ->double.

14.Method Overriding:(Run-time polymorphism):

(i)Whatever method available in the parent class,by default available to the child class through inheritance.

**(ii)If the child class not satisfied to the parent class method implementation,then that particular method
child class is redefined.This concept is nothing but "Method Overriding".

(iii)The Parent class method which is overridden is called "overridden method" and child class method
which is overriding is called "overriding method".

**(iv)Method resolution(which method gets a chance) always takes care of "JVM" based on the run time object.

Parent obj=new Child(); //Here new Child is a run time object.
Parent obj=new Parent(); //Here new Parent is a run time object.

(v)So it's called run-time /dynamic /late-binding polymorphism.

**(vi)Return type of method overriding must be the same as before jdk version 1.4 .
But from jdk 1.4 version onwards we can return different return type of overriding method is called "Covariant return type".

**And "Covariant return type" is only applicable only for Object type, but not for primitive type.

And the return type of the overriding method must be the child return type of overridden method return type.

Parent class type = Object ,Number
child class type of Object = String,StringBuffer,StringBuilder,Boolean.....
child class type of Number = Byte,Short,Integer,Long,Float,Double

(vii)Private method can't be override.(Because it is within the class only).
final method can't be override.(Bcz that is the final implementation)
Override process:-
final to non-final :-compiler error
non-final to final :-successfully execute.
abstract to non-abstract :- acceptable
non-abstract to abstract :-acceptable (abstract method will new implement in nextlevel child class)
Synchronize to non-synchronize And vice-versa is acceptable.
native to non-native And vice-versa is acceptable.
strictfp to non-strictfp And vice-versa is acceptable.

**(viii)While performing overriding we can't reduce the scope of the access modifier. (like public to private)
If we increase the scope of access modifier then no problem at all in override.
We can write the same modifier both in overriding and overriding acceptable except private.
i.e public > protected > default >private

**(ix)If child class method throws any "Checked Exception", then compulsory parent class method should
throw the same checked exception or it's parent type(Throwable / Exception).
For "Unchecked Exception" there is no restriction,you can use it anywhere in overriding.

#Throwable

1.Exception

(a)RuntimeException(AE,NPE,CCE)
(b)IOException (FNFE,EOFE)
(c)InterruptedException
(d)ServletException

2.Error

(a)VM Error
i.OutOfMemoryError
ii.StackOverFlowError

Checked Exception :- (IOException , InterruptedException ,ServletException)

Unchecked Exception :- (RuntimeException , VM Error)

(x)If parent class method is static and child class method is not static And Viceversa ,then it is not Overriding,
it will give a compile time error.
If both override and overriding method are static ,then method overriding is not possible But
"Method hiding is possible".
**(xi)If both the method is static then Method resolution(which method get chance) always takes care by "compiler" based on reference type.(this is method hiding).
Method Hiding is compile_time/static/early polymorphism.

(xii)If the argument type is not the same, then that is method overloading. (special case for var arg type(int...i))
If in both methods the argument type is var arg(int...i), then it's method overriding.

**(xiii)Variable resolution(which variable to be called) always takes care by "compiler" based on reference type.
This is called "variable hiding" OR "Shadowing" .
instance variable to instance variable
static variable to non-static variable
non-static variable to static variable
static variable to static variable
(All above variable resolution based on reference type)
**That means overriding concepts applicable for method not for variable.

15.Polymorphism:

(i)Polymorphism is a Greek word.Poly means "many" and morphs means "form".
(ii) One name(method) but multiple forms is called polymorphism.
(iii) There are two types of polymorphism:-
(a)compile time polymorphism / Static / Early binding
i.Method Overloading
ii.Method Hiding (if you override static to static method)
(b)Run time polymorphism / Dynamic / Late binding
i.Method Overriding

- The main 3 pillars of OOPs concept is :
  (a)Encapsulation : Security
  (b)Inheritance : Reusability
  (c)Polymorphism : Flexibility

16.Constructor:

(i)Constructor is mainly used to perform initialisation of an Object.
And the name of the constructor must be the same name as the class name.

**(ii)When we create an object using keyword "new" ,then the instance variable value by default
null takes care by JVM. After the "new" constructor will execute.
After constructor calling the value of argument will pass to the constructor ,And then we can use
"this" keyword for initialisation of the object instance variable.
(iii)Number of object creation ===== number of construction execution.
**(iv)JVM calls the constructor automatically when an object is created,But we are not calling the constructor.
so we have not to expect a return type from the constructor.
**(v)We can use only public,default,protected,private access modifiers in Constructor.
**(vi)If we do not create any constructor,then "compiler" creates a default constructor automatically.
If we have at least one constructor also then the compiler does not create Default constructor.
That's why every class in java should compulsorily contain constructor.
Default constructor:
(i)It's always no argument constructor.
(ii)Access modifier of default constructor same as class modifier(only public ,default).
(iii) Inside the Default constructor there is only one line of code. i.e super();
Here super(); is a no argument call to super class constructor.
(vii)Inside the constructor we can use the "this" or "super" keyword .
But "super" keyword always be the first statement in the constructor.
If we use super and this simultaneously then it gives a compile time error.
(super(); means call super class constructor , this(); means call current class constructor)
(viii)Overriding and Inheritance are not applicable for constructor.
(Because parent constructor is not available to child constructor in Constructor concept)
(ix)Constructor concept is not applicable for interfaces.
(Bcz interface instance variable only public,static,final.
So here no possible to initialize the instance variable.That's why no need to use constructor in interface).

17.String ,StringBuffer and StringBuilder:

(i)String is immutable but StringBuffer is mutable.
For adding some contents in String we use concat() method in String ,But we have to store the changing contents
to another String reference variable.
But in StringBuilder there is no need to create new StringBuilder reference variable to store the adding contents
by using append() method.

**(ii)The parent class of String,StringBuffer and StringBuilder is Object class.

In Object class the "==" and ".equals()" methods are for reference/address comparison.

But String is overridden with Object Class ,so it's .equals() used for content comparison.

But StringBuilder is not overridden with the Object class,so it's .equals() used for reference/address comparison.

(iii) String s=new String("Nayak");

Here after using new keyword the content "Nayak" store in "Heap" memory area and also a copy of String("Nayak")

will store in SCP(String constant pole/Literal) for future re-use of this string.(Here 2 object created)

String s="Nayak";

Here only String content value store in SCP/literal memory area.(Here 1 object created)

(iv) String s1=new String("Nayak");
String s2=new String("Nayak");
String s3="Nayak";
String s4="Nayak";

Here 1st s1 object will be created in heap area because of the new keyword and also an object copy created in SCP/literal.

After s1, s2 objects will be created in the heap area again ,But not scp because "Nayak" is present already in SCP/literal.

Then s3 and s4 not creates object again because same content already in SCP area.

Heap	SCP/Literal
s1->"Nayak"	"Nayak"
s2->"Nayak"	s1,s2,s3,s4

That means here only 3 times object will created.

And concat(); , append(); String contents always 1st store in SCP / String Literal and after store in Heap Area.

(v)String is immutable , so the memory management is good and system not behave abnormal condition.

And there is no need to create object again and agian for same content in SCP/literal area.

Best Example: Hyderabad Voting 1cr voters city name same.

If a voter wants to change city name Hyderabad to vijaybada then,automatic an another object will be created for vijayawada without changing the city name Hyderabad because of beauty of immutable concept of String in JAVA.

**(vi)All wrapper class objects are by default "immutable" as like String.

(vii)If StringBuffer is final ,then also mutable properties is no loss.

If "final" used in StringBuffer then we are not able to change the reference.(means storing in another object)

Example:
```
final StringBuffer s=new StringBuffer("Hello");
s.append(" World");
s=new StringBuffer(" World");//Compiler error, due to final re-assigned to other not possible.
System.out.println(s);//Hello World
```

(viii)Default capacity of StringBuffer is 16. (use method .capacity() for knowing capacity)
If we use less than 16,then capacity= 16 + no of char
i.e. StringBuffer s=new StringBuffer("abcdefghijklm"); //s=16+13=29
If we use more than 16 ,then capacity= (no. of char +1)*2
We can also set the capacity of StringBuffer i.e StringBuffer s=new StringBuffer(1000);
If we want more than 1000 capacity ,then automatically capacity=(1000+1)*2 .

******StringBuffer sb=new StringBuffer(int initialCapacity);******
ImpStringBuffer sb=new StringBuffer(String s);******
Example:
StringBuffer sb= new StringBuffer("Shivani");
System.out.println(sb.capacity()); ////23 Explanations : default capacity +no.of string
: 16+07=23

(ix)StringBuffer class method:-
length(); chartAt(int); setCharAt(int,char); append(any type of data can add);
insert(index pos,any type); delete(int begin,int end); deleteCharAt(int index);
** reverse(); setLength(int); ensureCapacity(int); trimToSize();
{command : javap java.lang.StringBuffer }

(x)In StringBuilder multiple threading concepts are allowed.But in StringBuffer one threading is allowed at a time.
StringBuilder is synchronized ,But StringBuffer is Synchronised.
The methods which are applicable in StringBuffer ,these all are applicable in StringBuilder.
(xi)"Method chaining" is that ,return type of all must be method same(may be String,StringBuffer,StringBuilder)
sb.m1().m1().m3().m4().....
ie. StringBuilder sb=new StringBuilder();
sb.append("Sharma").reverse().append("Nayak").insert().....like this.

18.Difference between StringBuffer and StringBuilder??

StringBuffer	StringBuilder
Every Method Present In Stringbuffer Is Synchronized.	No Method Present In Stringbuilder Is Synchronized.
At A Time Only One Thread Is Allow To Operate On Stringbuffer Object And Hence It Is Thread Safe.	At A Time Multiple Thread Are Allowed To Operate On Stringbuilder Object And Hence It Is Not Thread Safe.
Threads Are Required To Wait To Operate On Stringbuffer Object And Hence Relatively Performance Is Slow.	Threads Are Not Required To Wait To Operate On Stringbuilder Object And Hence Relatively Performance Is High.
Introduced In 1.0 Version.	Introduced In 1.5 Version.

19.Exception Handling:

(i)Unwanted or unexpected event that disturbs the normal flow of program,is called "Exception".Always in run time only.

(ii)Gracefull termination of the program called exception handling.(means no data loss if any problem/error in program)

(iii)Defining alternative way to continue rest of the program normally,is called "Exception Handling".

**(iv)Inside a method if an exception raised,the method in which it raised is responsible to create exception object
by the help of JVM internally.If the exception is not handled in the method,then method will terminated.
So this exception is handled by JVM using "default exception handler".This is abnormally termination
but not graceful termination.

(v)The root of exception heirarchy is "Throwable" and it's having two child classes these are:

#Throwable

1.Exception 2.Error

(a)RuntimeException(AE,NPE,CCE,IOFE{AIOBF,SIOBF},IAF{NFE}) (a)VM Error
(b)LinkageError
(b)IOException (FNFE,EOFE) i.OutOfMemoryError
i.VerifyError
(c)InterruptedException ii.StackOverFlowError
(d)RemoteException

(e)ServletException

CCE:-ClassCastException ,
IOFE:-IndexOutOfBoundException,
IAF:-IllegalArgumentException
NFE:-NumberFormatException

Checked Exception :- (Throwable,Exception,IOException ,InterruptedException ,ServletException)

Unchecked Exception :- (RuntimeException , VM Error)

(vi)The exception which are checked by compiler for smooth execution of the program at runtime,whether the programmer handling or not, such type of checking is done by "compiler".This type of things is called "Checked Exception".

(vii)The exception which are not checked by "compiler" is called Unchecked Exception.(like 10/0)

(viii)If the parent exception is checked but child exception need not to check,these type of exception is called "partially checked exception".
ie. Exception , Throwable both 2 are partially checked exception.
But if parent exception and child exception both must to be check is called "fully checked exception"

(ix)Every java program have a compulsory thread ,that is main() thread.

**(x)If there is no exception ,then catch block will not to be execute.
Inside try block if any statement exception raised even though you handled that exception in catch block, rest of the try block is not gone to be executed.

(xi)Methods for print Exception:
1)e.printStackTrace();
2)System.out.println(e); or System.out.println(e.toString());
3)System.out.println(e.getMessage());
//These 3 methods are available in Throwable class,so all 3types methods can use in any type of Exception.

(xii)If there are different type of exception in try block,then we have to write separate catch block for handling every exception and then write default catch block.This is the good habit of best programmer.
//(not write like catch(Exception e)) means only in one catch block,all exceptions are handled.
And always write exception in catch block from child to parent class type exception,Because JVM always check the catch block from top to bottom.

(xiii)
```
try{
    //Risky code
```

```
        }
    catch{
        //Handling Code
       }
    finally{
      //Clean Up Code
      }
```

(xiv) If there is no exception in try block,then also finally block will executed takes care by JVM.

If we write "System.exist(0);" then JVM will shutdown explicitly.Then finally block will not execute.

```
//System.exit(0) automate happen when current will gone.
   here argument means "status code"
    System.exit(0);//zero means normal termination
    System.exit(5);//non-zero means abnormal termination
```

(xv)If there is much more risky code in try block,then we have to use nested try/catch block for excute all line of code of try block.

(xvi)The order of try,catch ,finaaly is try->catch->finally.

If we write anything in between try and catch or catch and finally block then we will get compile time error.

Example:

```
try{
    System.out.println("Hello");
   }
   System.out.println("Hi"); //try without catch compile time error
catch(Exception e){
   System.out.println(e.getMessage());//catch without try compile time error
   }
  System.out.println("Bye");
finally{
   System.out.println("It's Good"); //finally without try block,compile time error.
   }
```

throw keyword:-

**(xvii)To handover our created exception object to the JVM manually,so we use "throw" keyword.

Our own exception is also known as Customized Exception.

The programmer is throw the exception object by "throw" keyword and handover to the JVM mannualy,

that exception is catch by the JVM.

i.e. throw new InsuffientFundException

//here "new" is used to create the exception object.

(xviii)After "throw" we can't write any statement directle.

i.e throw new Exception;

System.out.println("Hello") //CE, unreachable statement

throws keyword:-

(xix)We can use "throws" to deligate responsibility of exception handling to the caller(JVM /another method).

**It's required only used for checked exception,But foe unchecked exceptions there is no use.

But it's some time give abnormal termination of program.

(xx)If there is no exception in try block,then no need to write fully checked exception in catch block.

But if you write uncheked exception and partially checked exception in catch block,then there is no problem.

i.e.
```
try{
  System.out.println("Hello");
   }
catch(IOException){
   }     //compiler error we will get.
```

**(xxi)Garbage collector will call "finalize()" method to perform clean up activities.

(example:-non reference object is food fo GC)

//finalize() will close the DB/Network connection of an non reference object,before goes to the Garbage collector.

(xxii)The exceptions which are defined explicitly by programmer to meet programming required

are called "user defined exception".

(xxiii)We are no need to close resources explicitly,when we use try with resources.so finally block is not required.

20.Multi Threading:

(i)Multi Tasking:

Executing several task simultaneosly,is a concept of multitasking.

It is used to reduce responce time of CPU and to improve perfermance.

There are 2 type of multitasking,these are:-

(a)Process Based multitasking:(OS level)

Executing several task simultaneosly where each task is a separate independent process.

(Proceses are not depend with another process)

(b)Thread based multitasking:(Programming level)

Executing several task simultaneously where each task is a separate independent part of

the same program,is called thread based multitasking.

(ii)Thread:

Flow of execution of the program is called "Thread".

We can define a thread in the following two ways-

(i)By extending thread class

(ii)By implementing runnable interface

(i)By extending thread class:(in this process we are missing inheritance approach)

*Example is available in my eclipse

When we write t.start(); then a new thread will created and it's responsible for execution
of run() method.
If start() method is not present in child class of Thread class,then start() method executed from Thread class.
**It is highly recommended to override run() method,otherwise don't go through the multi threading concept.
**Thread class start() method is called the heart of multi-threading.
**start() method only calls no-arguments run() method only ,when run() is overloaded.

**"Thread scheduler" is the part of JVM and it is responsible to schedule threads in an order.
We can't expect exact algorithm followed by thread scheduler.(order is not same always)
It varies from JVM to JVM.
Hence We can't expect thread execution order and exact output.

**life cycle of thread:-

t.start() allocate processor run()
new/born ---------->ready/runnable---------->running-------->dead
(MyThread t=new MyThread,thread object instantiation)

After starting a thread(t.start()) if we are trying to restart the same thread once again, then "IllegalThreadStateException" will get.

(ii)By implementing runnable interface:(this process is recommended)
*We can define a thread by implementing a runnable interface.
*runnable interface present in java.lang package and it contains only one method i.e. run().
This process is recommended for programmers because we can extend and implement the classes.
i.e class MyRunnable extends ABC implements Runnable
But in by extending Thread class process there is no possible to extends the class after
extending the Thread class.(multiple Inheritance not supported)

(ii)Thread Priorities:(1 to 10)

* Valid range of thread priority is 1 to 10.
* Here 1 is "min priority" and 10 is "max priority". And normal priority is 5.

Thread.MIN_Priority
Thread.MAX_Priority
Thread.NORM_Priority

* If two thread having same priority,then we can't expect exact execution order.It depends on thread scheduler.
* Here for priority there are two method these are,
public final int getPriority()
public final void setPriority(int p)
If we write t.setPriority(11) ,then we get CE i.e "IllegalArgumentException".
** The default priority only for the "main" thread is 5.But for all remaining threads default priority
will be inherited from parent to child.
ie. Whatever priority parent thread has, the same priority will be there for the child thread.

(iii)Prevent a Thread Execution:
* We can prevent a thread execution by using the following methods:-
. yield()
. join()
. sleep()

yield() method:
yeild() method causes to pass current executing thread to give the chance for waiting threads
of same priority.
If there is no waiting thread or all waiting threads have low priority, then same thread can
continue it's execution.
If multiple threads are waiting with same priority then which waiting thread will get the chance
we can't expect .It depends on thread scheduler.
Example:-Land line Telephone
prototype of yield() method:-
pulic static native void yield();
** //The method which are not implemented in java .These are implemented in non-JAVA.

join() method:
If a thread wants to wait until completing some other thread then we should go for join() method.
For example if a thread t1 wants to wait until completing t2,then t1 has to call t2.join() .
prototype of yield() method:-
pulic final void join();
pulic final void join(long ms); //if a thread wants to wait for a speific time.(join with time)
pulic final void join(long ms,int ns); //ms-mili sec ,ns-nano sec
Every join() method throws InterruptedException because in the time of waiting may be other thread

will come and interrpted the waiting thread ,by interrupt() method in Thread..
If main thread is join() then excution will be stocked .
i.e Thread.currentThread().join();

sleep() method:
If a thread don't want to perform any operation for a particular amount of time then we should go for sleep() method
prototype of sleep() method:-
pulic static native void sleep(long ms);
public static void sleep(long ms,int ns); //it's not native method
Every sleep() method throws InterruptedException because when sleep some other thread
may interrupted, by interrupt() method in Thread.

(iv)Synchronization:
* Synchronized is the modifier applicable only for the methods and blocks but not for classes and variable.
* If multiple threads are trying to operate simultaneously on the same java "object" then there may be a chance of
"Data inconsistency problem".(Ex-train ticket booking at a time)
**To overcome this problem we should go for synchronized keyword.
*If a method or block declared as synchronized then at a time only one thread is allow to execute,
that method or block on the given object and after 1st thread execution completed then 2nd one will start execution.So that data inconsistency problem will be resolved.
*The main advantage of synchronized keyword is we can resolve data inconsistency problems, But the main
disadvantage of synchronized keyword is "it increases waiting time of threads and creates performance problems".
Hence if there is no specific requirement,then it is not recommended to use synchronized keyword.
Example:-lock key concept for synchronized(object is key)
** If very few lines of code required synchronization then it is not recommended to declare entire
method as synchronized.We have to enclose those few lines of the code by using "synchronized Block".
Main advantage of synchronized Block is it's reduce the waiting time as compare to synchronized method.
** We can declare synchronized Block as follows:-
To get a lock of current object-

```
synchronized(this)
{
}
```

To get a lock of particular object 'b'-

```
synchronized(b)
  {
    }
```

To get a class level lock-

```
synchronized(Display.class)  //Here Display is the class name.
 {
    }
```

*Inter thread communication:

Two threads can communicate with each other by using wait(); notify() ; and notifyAll() methods.

The thread which is expecting updation is responsible to call wait() method, then immediately the thread entered into waiting state.

The thread which is responsible to perform updation, after performing updation it is responsible to call notify() method then waiting thread will get the notification and continue it's execution with those updated items.

All these 3 methods wait(); notify() ; and notifyAll() are available in "Object class" but not in "Thread class".

Prototype of wait(),notify(),notifyAll() method:-

```
public final void wait() throws IE;
public final native void wait(long ms) throws IE;
public final void wait(long ms,int ns) throws IE;

public final native void notify(long ms);
public final native void notifyAll(long ms,int ns) //these two are not wait so IE is not throws here.
```

Why can't we override the static method?

It is because the static method is bound with class whereas the instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

SUMMARY OF PUBLIC , PROTECTED , DEFAULT AND PRIVATE

Visibility	Public	Protected	Default	Private
With in the same class	Yes	Yes	Yes	Yes
From child class of same package	Yes	Yes	Yes	No
From non-child class of same package	Yes	Yes	Yes	No
From child class of outside package	Yes	Yes (we should use childreferences only)	NO	No
From non-child class of outside package	Yes	No	No	No