

**Snippet ::**

**1.**

**pubilc class TestApp{**

**public static void main(String... args){**

**StringBuilder sb=new StringBuilder(100);**

**System.out.println(sb.length() + ":"+sb.toString().length());**

**}**

**}**

A. 100:100

B. 100:0

C. 16:16

D. 16:0

E. 0:0

F. CompileTimeError

G. NullPointerException

**2.**

**pubilc class TestApp{**

**public static void main(String... args){**

**StringBuilder sb=new StringBuilder();**

**System.out.println(sb.append("").append("").append("").length());**

**}**

**}**

A. CompileTimeError

B. NullPointerException

C. 0

D. 3

E. None of the above

**3.**

**pubilc class TestApp{**

**public static void main(String... args){**

**StringBuilder sb=new StringBuilder();**

**System.out.println(sb.append(null).length());**

**}**

**}**

A. CompileTimeError

B. NullPointerException

C. 0

D. 4

E. None of the above

**4.**

**pubilc class TestApp{**

**public static void main(String... args){**

**String s= " ";**

**boolean isEmpty= s.isEmpty();**

**System.out.println(isEmpty);**

**s.trim();**

**isEmpty= s.isEmpty();**

**System.out.println(isEmpty);**

**}**

**}**

A. CE

B. NullpointerException

C. true

true

D. false

false

E. true

false

F. false

true

G. None of the above

**5.**

**pubilc class TestApp{**

**public static void main(String... args){**

**StringBuilder sb=new StringBuilder("RahulDravid");**

**sb.delete(5,6).append(5," D").toString().toUpperCase();**

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

**}**

**}**

A. RAHUL DRAVID

B. RAHUL DAVID

C. Rahul Dravid

D. Rahul David

E. CE

F. StringIndexOutOfBoundsExeption

**Exception continuation**

**======================**

**Note::** finally block is meant for closing the resources(clean up activity) associated with try block.

finalize() block is meant for cleanup activity assoicated with the object.

We prefer writing finally block over finalize() method because we can't expect the behaviour of GC.

**Control flow in try catch and finally**

**=====================================**

**try{**

**statement-1**

**statement-2**

**statement-3**

**}catch(Exception e){**

**statement-4**

**}finally{**

**statement-5**

**}**

**statement-6**

Case1: **If there is no exception.**

statement => 1,2,3,5,6 normal termination.

Case2: **If an exception is raised at statement2 and corresponding catch block is matched.**

statement => 1,,4,5,6 normal termination.

Case3**: If an exception is raised at statement2 and corresponding catch block is not**

**matched.**

statement => 1,5(finally block) abnormal termination

Case4: **If an exception is raised at statement4.**

statement => 1,2,3,5, abnormal termination

Case5: **If an exception is raised at statement5.**

statement =>1,2,3 abnormal termination

**Control flow in Nested try-catch-finally**

**========================================**

**try{**

**stmt-1**

**stmt-2**

**stmt-3**

**try{**

**stmt-4;**

**stmt-5;**

**stmt-6;**

**}catch(X e){**

**stmt-7;**

**}finally{**

**stmt-8;**

**}**

**stmt-9;**

**}catch(Y e){**

**stmt-10;**

**}finally{**

**stmt-11;**

**}**

**stmt-12;**

Case1: **If there is no exception.**

statement => 1,2,3,4,5,6,8,9,11,12 normal termination.

Case2: **If an exception is raised at statement2 and corresponding catch block is matched.**

statement => 1,10,11,12 normal termination.

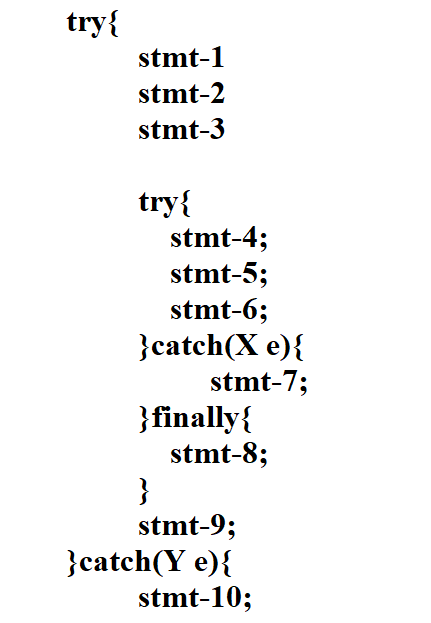
Case3: **If an exception is raised at statement2 and corresponding catch block is not**

**matched.**

statement => 1,11 abnormal termination.

**Control flow in Nested try-catch-finally**

**========================================**

****

Case5: **If an exception is raised at statement5 and inner catch has not matched but outer**

**Catch block is matched.**

Statement =>1,2,3,4,8,10,11,12 normal termination

Case6: **If an exception is raised at statement5 and both inner catch and outer**

**catch block is not matched.**

Stateent => 1,2,3,4,8,11 abnormal termination

Case7: **If an exception is raised at statement7 and corresponding catch block is**

**matched.**

Statement =>1,2,3,8,10,11,12 normal termination

Case8: **If an exception is raised at statement7 and corresponding catch block is**

**not matched.**

Statement => 1,2,3,8,11 abnormal termination

Case9: **If an exception is raised at statement8 and corresponding catch block is matched. .** Statement => 1,2,3,4,5,6,10,11,12 normal termination.

Case10: **If an exception is raised at statement8 and corresponding catch block is not**

**matched.**

Statement => 1,2,3,4,5,6,11 abnormal termination.

Case11: **If an exception is raised at statement 9 and corresponding catch block is**

**matched.**

Statement => 1,2,3,4,5,6,8,10,11,12 normal termination

Case12: **If an exception is raised at statement 9 and corresponding catch block is**

**not matched**.

Statement => 1,2,3,4,5,6,8,11 abnormal termination.

Case13: **If an exception is raised at statement 10.**

Statement => abnormal termination.

Case14: **If an exception is raised at statement 11 or 12.**

Statement => abnormal termination.

**Note::**

**1. If we are not entering into try block then finally block wont be executed.**

**2. If we enter into try block , there is every possibility of finally block getting**

**executed.**

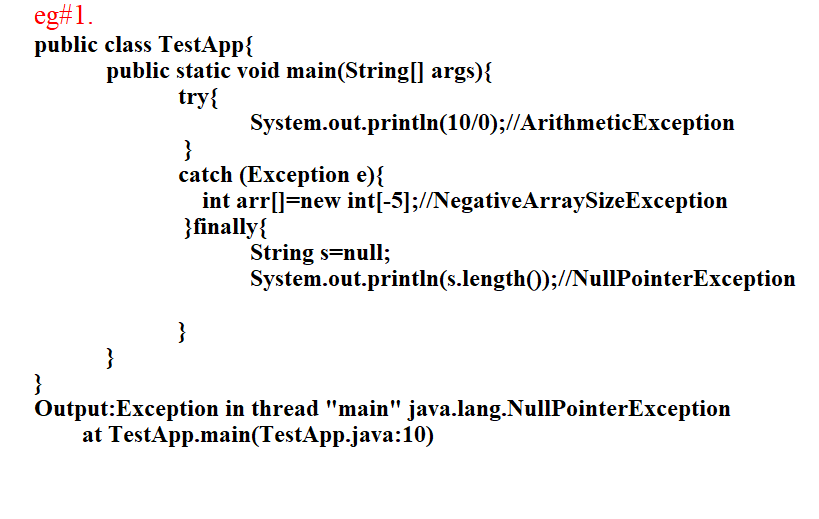
**3. Nested try catch is possible.**

**4. Normally in the code, we always keep specific catch block inside the inner**

**try block whereas Generic catch block is been kept at the outer catch**

**block.**

**DefaultExceptionHandler will handle only one exception and that to the exception which has occurred recently, not all the exceptions which has occured in the program.**



**Various possible cases of writing try catch and finally block**

**===========================================**

**1.try{**

**}catch(X e){ //valid**

**}**

**2. try{**

**}catch(X e){ //X must be child whereas Y must be parent only then valid.**

**}catch(Y e){**

**}**

**3. try{**

**}catch( X e){ //CE: Exception is already been caught**

**}catch( X e){**

**}**

**4. try{**

**}finally{ //valid combination(we dont' worry about abnormal**

**termination)**

**}**

**5. try{**

**}catch(X e){ //valid combination**

**}finally{**

**}**

**6. try{} //CE: try without catch, finally**

**7. catch(){} //CE: catch without try**

**8. finally{} //CE: finally without try**

**9. try{}**

**System.out.println("Hello"); //CE:catch without try**

**catch(){}**

**10. try{}**

**catch(X e){}**

**System.out.println("hello"); //CE:catch without try**

**catch(Y e){}**

**11. try{}**

**catch(X e){}**

**System.out.println("hello"); //CE: finally without try**

**finally{}**

**12. try{}**

**finally{} //CE:: catch without try**

**catch(X e){}**

**13. try{}**

**catch(X e){} //valid**

**try{}**

**finally{}**

**14. try{}**

**catch(X e){}**

**finally{}**

**finally{} //CE: finally without try**

**15. try{}**

**catch(X e){**

**try{}**

**catch(Y e1){} //valid**

**}**

**16. try{}**

**catch(X e){}**

**finally{**

**try{}**

**catch(Y e1){} //valid**

**finally{}**

**}**

**17. try{**

**try{} //CE: try without catch,finally**

**}**

**18. try**

**System.out.println("hello"); //CE**

**catch(X e){}**

**19. try{}**

**catch( X e1)**

**System.out.println("hello"); // CE**

**20. try{}**

**catch( NullPointerException e1){}**

**finally //CE**

**System.out.println("Hello");**

**Note::**

* Whenever we are try block,compulsorily we should write either catch or

finally.

* catch without try and finally without try is invalid.
* Whenenver we write try catch and finally order is important,

(ex try,catch,finally).

* try with catch, and try with finally is valid combination
* Nesting of try catch finally is permitted.
* for try,catch,finally curly braces are mandatory.

**Usage of throw keyword**

**======================**

It is basically used to throw the exception object manually to jvm by the user.

**Syntax:: throw Exceptionobject.**

**throw=>** To handover the exception object manually to the jvm we use throw keyword.

**ExceptionObject =>** creating the exception object manually using new keyword with suitable message.

eg#1.

**public class TestApp{**

**public static void main(String[] args){**

**System.out.println(10/0);**

**}**

**}**

Explanation:: Creation of ArithmeticException and handover the exception object to jvm will be done by main() automatically.

eg#2.

**public class TestApp{**

**public static void main(String[] args){**

**throw new ArithmeticException("/ by zero");**

**}**

**}**

Explanation::Creation of ArithmeticException and handover the exception object to jvm will be done manually.

**CaseStudies**

**===========**

case1:: **throw e;**

=> if 'e' refers to any exception object then we will get the details of

Exception object.

=> if 'e' referes to null, we get NullPointerException.

eg#1.

**public class TestApp{**

**static ArithmeticException e=new ArithmeticException("/by zero");**

**public static void main(String[] args){**

**throw e;**

**}**

**}**

Exception in thread "main" java.lang.ArithmeticException: /by zero

at TestApp.<clinit>(TestApp.java:2)

vs

**public class TestApp{**

**static ArithmeticException e;**

**public static void main(String[] args){**

**throw e;**

**}**

**}**

Exception in thread "main" java.lang.NullPointerException

at TestApp.main(TestApp.java:5)

case2::After throw statement,we can't write any statements,if we try to write it would result in compile time error saying "unreachable code".

eg#1.

**public class TestApp**

**public static void main(String[] args){**

**System.out.println(10/0);**

**System.out.println("hiee");**

**}**

**}**

VS

**public class TestApp{**

**public static void main(String[] args){**

**throw new ArithmeticException("/by zero");**

**System.out.println("hiee");**

**}**

**}**

case3::To throw any object using throw keyword, we need to make sure that object is directly or indirectly related to "Throwable" class otherwise it would result in "CompileTimeError".

**public class TestApp{**

**public static void main(String[] args){**

**throw new TestApp();**

**}**

**}**

Output::Compile Time Error:: incompatible types TestApp can't be converted to "Throwable".

VS

**public class TestApp extends RuntimeException{**

**public static void main(String[] args){**

**throw new TestApp();**

**}**

**}**

Output::Compilation succefull.

**Snippets**

**========**

**public class TestApp{**

**public static void main(String... args){**

**String text="RISE";**

**text+=(text=" ABOVE");//line n1**

**System.out.println(text);**

**}**

**}**

A. RISE ABOVE

B. ABOVE ABOVE

C. RISE RISE ABOVE

D. Compile Time Error at line n1

E. NullPointerException

**2. public class TestApp{**

**public static void main(String... args){**

**String s="PANIC";**

**StringBuilder sb=new StringBuilder("THET");**

**s=s.replace("N",sb);**

**System.out.println(s);**

**}**

**}**

A. PANIC

B. PATHETIC

C. CompileTimeError

D. PANTHETIC

E. StringIndexOutOfBoundsException

"throw" keyword is also one of the way of handling the exception,but it won't guarantee that it would result in "Normal Termination/smooth termination".

=>throw keyword is normally prefered when we work with "Custom Exceptions".

**throws**

**=====**

* It is a keyword which is basically meant for handling the exception.
* It is mainly used when the exception type is "Checked".
* If the exception checked then at the method prototype, we can keep throws

keyword through which we can inform the compiler that exception is

thrown.

* throws keyword is basically used to delegate the exception object to the caller.

eg::

**public class TestApp {**

**public static void main(String[] args){**

**PrintWriter pw=new PrintWriter("abc.txt");**

**pw.println("Hello ");**

**}**

**}**

output::

CompileTimeerror::unreported exception FileNotFoundException; must be caught or declared to be thrown

eg:

**public class TestApp {**

**public static void main(String[] args){**

**Thread.sleep(2000);**

**}**

**}**

output::

CompileTime error: unreported exception InterruptedException; must be caught or declared to be thrown Thread.sleep(10);

Since the exception is checked we can just delegate the exception to the caller to handle using throws keyword.

eg#1.

**public class TestApp {**

**public static void main(String[] args)throws InteruptedException{**

**Thread.sleep(10);**

**}**

**}**

**usage of throws keyword in realtime**

**===================================**

**public class TestApp {**

**public static void main(String[] args)throws InterruptedException{**

**System.out.println("Hello");**

**doWork();**

**System.out.println("bye");**

**}**

**public static void doWork()throws InterruptedException{**

**doMoreWork();**

**}**

**public static void doMoreWork() throws InterruptedException{**

**Thread.sleep(1000);**

**}**

**}**

Output:: Hiee

Byee

**Behind the scenes**

=> jvm called main(),main() internally called doWork(), doWork() internally called doMoreWork() and the process continues

=> As noticed in doMoreWork(), their is a possibility of CheckedExeption called "InterruptedException", since it is checked compiler will check whether the method has handled or not through "try catch" or through throws.

=> if the method had throws keyword then in the method hierarchy all the methods should have handling code through "try catch" or through throws.

=> even if one of the method failed to do so it would result in "CompileTimeError".

**MainObjective of "throws" keyword is to delegate the exception object to the caller from the called method and that to the exception type would be normally "CheckedException".**

How many ways of Exception handling mechansim are availabe in java?

**a. try catch**

**b. throw**

**c. throws**

Does all the exception handling mechanism would promote smoothful termination of program?

Ans. No only try catch would promote that too if the exception is correctly handled in catch block.

**Different cases of throws keyword**

**=================================**

**case1::**

throws keyword can be used only on throwable type object, if we use on any other type it would result in compile time error.

eg#1.

**public class TestApp extends RuntimeException{**

**public static void main(String[] args) throws TestApp{**

**System.out.println("Hello");**

**}**

**}**

Output:: Compile time Error:TestApp is not of Throwable type.

vs

**public class TestApp extends RuntimeException{**

**public static void main(String[] args) throws TestApp{**

**System.out.println("Hello");**

**}**

**}**

Output:: Hello

**case2::**

If the exception type is unchecked then no need of throws in the method prototype,by default exeption object will be propoagated at the run time,if it is checked type then compulsorily we need to handle it through "try catch" or through "throws",but prefered is "throws" becoz it is "checked type".

**public class TestApp {**

**public static void main(String[] args) {**

**throw new Exception();**

**}**

**}**

Output:: Compilation Error.

To resolve this problem we use throws keyword in the method prototype.

VS

**public class TestApp {**

**public static void main(String[] args) {**

**throw new Error();**

**}**

**}**

Output:: Compilation succesfull

Note:: Unchecked Exceptions by default will have Propogation nature,whereas checkedExceptions will get the behaviour only if we use throws method in the method prototype.

**case3::**

In our try block if their is no chance of rising an exception then we should not write the corresponding catch block,if we write it would result in "CompileTime

Error" saying Exception XXX is never thrown in the body of corresponding try statement.

This rule is applicable only for "fully checked Exception" class.

eg#1.

**import java.sql.SQLException;**

**public class TestApp {**

**public static void main(String[] args) {**

**try**

**{**

**}**

**catch (java.io.IOException e)**

**{**

**}**

**}**

**}**

Output:: Compile Time Error

exception IOException is never thrown in body of corresponding try statement

eg#2.

**public class TestApp {**

**public static void main(String[] args) {**

**try**

**{**

**}**

**catch (java.sql.SQLException e)**

**{**

**}**

**}**

**}**

Output:: CompileTimeError

exception SQLxception is never thrown in body of corresponding try statement

eg#3.

**public class TestApp {**

**public static void main(String[] args) {**

**try**

**{**

**}**

**catch (Exception e)**

**{**

**}**

**}**

**}**

Output:: Compilation succesfull

eg#4.

**public class TestApp {**

**public static void main(String[] args) {**

**try**

**{**

**}**

**catch (ArithmeticException e)**

**{**

**}**

**}**

**}**

Output::CompilationSuccesfull

**case 5::**

throws keyword can be used only at method level and at constructor level, if we try to use at any other places then it would result in "CompileTimeError".

eg#1.

**public class TestApp throws Exception {**

**public TestApp()throws Exception{**

**}**

**void display() throws Exception{**

**}**

**public static void main(String[] args) throws Exception {**

**}**

**}**

Output:: Compile Time Error

TestApp.java:2: error: '{' expected

**Exception handling mechanism**

**============================**

a. **throw** => for custom exception to throw

b. **throws** => for exception propogation/delgation

c. **try catch** => for handling the exception.

**Keywords related to Exception handling**

**======================================**

1. **try** => risky code

2. **catch** => handling code

3. **finally** => resource releasing code

4. **throw** => manually throw an exception object to the caller (used for

CustomException)

5. **throws** => used for Exception Propogration.(used for CheckedException)

**Common Compile time errors in Exception handling**

**================================================**

1. Exception XXXX has been already caught

2. unreported Exception XXXX must be caught or declared to be thrown

3. Exception XXXX is never thrown in the body of try

4. try without catch,finally

5. catch without try

6. finally without try

7. incompatible type

required:Throwable

found:: Test

**Custom Exception**

**================**

**case1::**

int a[] =new int[3];

a[5]=10; => jvm executed the current line due to this execution it would result

in AIOBE

**case2::**

int a=10

int b=0;

int c=a/b; => jvm executed the current line due to this execution it would result

in A.E

**case3::**

java SampleApp Ten

int age;

age=Integer.parseInt(args[0]); //jvm executed the current line due to this

execution it would result in "NumberFormatException".

As noticed in the above cases, whenever certain events occurs if jvm can't execute the statements in succesfull way it internally generates the exception object based on the specific type with the help of internal coding.

**Rules followed to Generate a CustomException**

**============================================**

1. Write any class and make that class as Child for RuntimeException class(refer

exception hierarchy)

2. Write a constructor to get the required message for the cause.

3. use throw keyword to manually generate an exception and throw an exception

object to the caller.

Note::

* for any Exception which internally gets generated the parent is

"Throwable".

* Exception and Error are child classes of "Throwable".
* RunTimeException and it subclasses,Error and its subclasses are

UncheckedException Remaining all classes are checkedException.

* Always as a good programming practise, we make our class to behave as a

Exception class by taking the help of inbuilt class called "Runtime

Exception", because RuntimeException is unchecked and by default it gets

the nature of "ExcpetionPropogation".

**Scenario**

**========**

**Write a code to manually generate an Exception called**

**a.** TooYoungAgeException : if the age enetered by the user is less than 18

**b.** TooOldAgeException : if the age entered by the user is greater than 60

otherwise issue DriverLicense through email id.

eg#1.

**class TooYoungAgeException extends RuntimeException{**

**TooYoungAgeException(String msg){**

**super(msg);**

**}**

**}**

**class TooOldAgeException extends RuntimeException{**

**TooOldAgeException(String msg){**

**super(msg);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) {**

**int age=Integer.parseInt(args[0]);**

**if (age<18){**

**throw new TooYoungAgeException("Age is too low, plz wait**

**for some time ....");**

**}**

**else if (age>60){**

**throw new TooOldAgeException("Age is too high, first**

**try to walk....");**

**}**

**else{**

**System.out.println("You will DL though your email**

**id.....");**

**}**

**}**

**}**

Output:

D:\SeptemberOnline\Day23>java TestApp 10

Exception in thread "main" TooYoungAgeException: Age is too low, plz wait for some time ....

at TestApp.main(TestApp.java:16)

D:\SeptemberOnline\Day23>java TestApp 65

Exception in thread "main" TooOldAgeException: Age is too high, first try to walk....

at TestApp.main(TestApp.java:19)

D:\SeptemberOnline\Day23>java TestApp 21

You will DL though your email id.....

**Top10Exceptions**

**===============**

Exceptions are raised based on the event occured,so we classify exceptions into

2 types

**a.** JVMException

**b.** ProgrammaticException

**JVMException**

Exceptions which are raised automatically by the jvm whenever certain events

occurs are termed as "JVMExceptions".

eg:: ArrayIndexOutOfBoundsException

NullPointerException

**ProgramaticExeption**

Exception which are raised by the programmer explicitly/ by the API Developer

based on the event occured such exceptions are refered as "Programatic

Exception".

eg:: NumberFormatException,IllegalArgumentException

Note::

**API** stands for ApplicationProgrammInterface

API => Collection of .classFiles .

Collection of .class files will be zipped as "JAR".

Java language itself is an API based language which we use it with the help of

jar called "rt.jar" available in lib folder of jdk installation.

**Top10Exceptions normally occurs in java coding**

**==============================================**

1.ArithmeticException

2.ArrayIndexOutOfBoundsException

3.NumberFormatException

4.IllegalArgumentException

5.NullPointerException

6.StackOverFlowError

7.ExceptionInitalizerError.

8.ClassCastException

9.IllegalStateException

10.NoClassDefFoundError.

Remaining topics are

JDK1.7 and higher version enhancements

Rules associated with overriding and constructor for Exception handling.

**ExceptionInInitializerError**

**====================**

This exception would be raised whenever we try to intialize the static variable

and if the intialisation fails or if there occurs any problem during the execution

of static block.

It is child class of Error which is unchecked exception.

eg#1.

**public class TestApp {**

**static int i =10/0;**

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

**}**

Output:: java.lang.ExceptionInInitalizerError.

eg#2.

**public class TestApp {**

**static String s=null;**

**static{**

**System.out.println(s.length());**

**}**

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

**}**

Output:: java.lang.ExceptionInInitalizerError.

**IllegalArgumentException**

**========================**

It is child class of RunTimeException which is unchecked type.

It is a programmatic exception which is generated by the API code.

eg#1

**public class TestApp {**

**public static void main(String[] args) {**

**Thread t=new Thread();**

**t.setPriority(10);//priority value should be with in 1 to 10**

**t.setPriority(100);**

**}**

**}**

Output:: IllegalArgumentException

**Categorise the following exception into JVM Exceptions and programaticException/API Exception**

**=================================================**

**JVMException**

=> NullPointerException

=> ArrayIndexOutOfBoundsException

=> ArithmeticException

=> ClassCastException

=> ExceptionInInitalizerError

=> StackOverFlowError

=> NoClassDefFoundError

=> NegativeArraySizeException

...

...

ProgrammaticException

=> NumberFormatException

=> IllegalArgumentException

=> IllegalStateException

=> AssertionError

Rules of Overriding w.r.t Exception

===================================

**Rule::**

While overriding the child class method, if the child class throws any "checked" exception then compulsorily the parent class method should throw the same exception or its parent exception.

The above rule is not applicable for "UncheckedException".

eg#1.

**class Parent{**

**public void methodOne(){**

**System.out.println("Hello");**

**}**

**}**

**class Child extends Parent{**

**@Override**

**public void methodOne()throws Exception{**

**System.out.println("byee");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) {**

**Parent p= new Child();**

**p.methodOne();**

**}**

**}**

Output::

TestApp.java:8: error: methodOne() in Child cannot override methodOne() in Parent

public void methodOne()throws Exception {System.out.println("byee");}

^

overridden method does not throw Exception

eg#2.

**class Parent{**

**public void methodOne()throws Exception{**

**System.out.println("Hello");}**

**}**

**class Child extends Parent{**

**@Override**

**public void methodOne()throws Exception {**

**System.out.println("byee");}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws Exception {**

**Parent p= new Child();**

**p.methodOne();**

**}**

**}**

Output:: Bye

Compiler:: methodOne() of child is throwing Exception which is checked so compulsorily the parent class methodOne() should throw the same Exception or its parent.

It also checks wheter the programmer has wrote a handling code or not for checked Exception.

JVM:: Depending on the runtime object, it internally binds the method call and it executes

eg#3.

class Parent{

public void methodOne()throws Exception{

System.out.println("Hello");}

}

class Child extends Parent{

@Override

public void methodOne() {

System.out.println("byee");}

}

public class TestApp {

public static void main(String[] args) throws Exception {

Parent p= new Child();

p.methodOne();

}

}

Output:: Byee

Compiler:: It will check only the method name if the parent is throwing the exception and whether it is handled or not.

JVM:: It will bind the method call based on runtime object.

**Predict the output**

**==================**

Rules w.r.t Overriding

=====================

parent: public void methodOne() throws Exception{}

child : public void methodOne()

Output:: It compiles.

parent: public void methodOne(){}

child : public void methodOne() throws Exception{}

Output:: CompileTimeError

parent: public void methodOne()throws Exception{}

child : public void methodOne()throws Exception{}

Output:: It compiles

parent: public void methodOne()throws IOException{}

child : public void methodOne()throws IOException{}

Output:: It compiles

parent: public void methodOne()throws IOException{}

child : public void methodOne()throws FileNotFoundException,EOFException{}

Output:: It compiles

parent: public void methodOne()throws IOException{}

child : public void methodOne()throws FileNotFoundException,InterruptedException{}

Output:: CompileTimeError

parent: public void methodOne()throws IOException{}

child : public void methodOne()throws FileNotFoundException,ArithmeticException{}

Output:: It compiles

parent: public void methodOne()

child : public void methodOne()throws ArithmeticException,NullPointerException,RuntimeException{}

Output:: It compiles

eg#4.

**import java.io.\*;**

**class Parent{**

**public void methodOne(){System.out.println("Hello");**

**}**

**}**

**class Child extends Parent{**

**@Override**

**public void methodOne() throws ArithmeticException,**

**NullPointerException,RuntimeException {**

**System.out.println("byee");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Parent p= new Child();**

**p.methodOne();**

**}**

**}**

Rule2:: If the parent class constructor throws any exception compulsorily the child class should throw the same exception or its parent otherwise it would result in "CompileTimeError".

eg#1.

**class Parent{**

**Parent() throws Exception{**

**//Zero argument constructor**

**}**

**}**

**class Child extends Parent{**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Parent p= new Child();**

**}**

**}**

Output:: CompileTime error: unreported exception Exception in default constructor

class Child extends Parent{

^

1 error

eg#2.

**class Parent{**

**Parent() throws Exception{**

**//Zero argument constructor**

**}**

**}**

**class Child extends Parent{**

**Child() throws Exception{**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws Exception{**

**Parent p= new Child();**

**}**

**}**

Output:: Compiles fine with no output

Compiler :: Since the zero argument constructor is throwing a checked exception from parent,compulsorily the child class zero argument constructor should throw the same exception or its parent type.

JVM :: It will just use the new operator to create the object and uses constructor to initialize the value of the object.

Difference b/w DefaultConstructor and ZeroArgument Constructor?

DefaultConstructor:: It will be inserted by the compiler which is always zero argument and the first line code will be always super().

protoype:: accessmodifer ClassName(){

super();

}

ZeroArgument Constructor:: It is the constructor written by the programmer with

throws keyword attached to it,if it is checked exception type.

prototype:: public className() throws XXXX{

}

or

className() or throws XXXX{

}

**Enhancement of Exception hanlding in higher version of JDK**

**==============================================**

1. **Try with Resources**

Till JDK1.6, we should normally write a coding of using the resource inside

try block if exception occurs write handling code inside catch block.

Inside finally block we should write the code or resource releasing which is

used inside try block.

It is as shown below.

eg#1.

**import java.io.\*;**

**public class TestApp {**

**public static void main(String[] args){**

**BufferedReader br=null;**

**try{**

//open the resource to read the contents of the file

**br=new BufferedReader(new FileReader("abc.txt"));**

**//use br to read the contents of the file**

**}**

**catch (IOException e){**

**e.printStackTrace();**

**}finally{**

**try{**

**if (br!=null)**

**{**

**br.close();**

**}**

**}**

**catch (IOException ie)**

**{**

**ie.printStackTrace();**

**}**

**}**

**}**

**}**

As noticed above, the lines of code written is more because we should explicitly write a code for resource releasing logic which acutally distrubs the readability

and also decreases the performance internally.

Solution:: Use Trywith resources and change the jdk version to 7 and above.

Whenever we use "try with resources", our focus will be only on the logic of opening the resource and using the resources, internally jvm will take care of closing the resource logic through "try with resource" syntax.

In try with resource,automatically the resource used in the try block will be closed once the control comes out of try with normal manner or in abnormal manner.

eg#1.

**import java.io.\*;**

**public class TestApp {**

**public static void main(String[] args){**

**try(BufferedReader br=new BufferedReader(new**

**FileReader("abc.txt"))){**

//use br to read the contents of the file

**}catch (IOException e){**

**e.printStackTrace();**

**}**

**}**

**}**

As noticed above, we used try with resources so the jvm will close BufferedReader resource as soon as the control comes out of try.

In this syntax,the lines of code is less through which it promotes readablity and it also increases the performance of the application.

In try with resource syntax what type of resources can be used?

=> While using try with resource syntax, we need to make sure the resources should implement an interface called "java.lang.AutoCloseable" interface,only those resources should be used otherwise it would result in "CompileTimeError".

In JDK1.7 "java.lang.AutoCloseable" interface was introduced.

**Note::**

**public interface java.lang.AutoCloseable {**

**public abstract void close() throws java.lang.Exception;**

**}**

Which ever classes has implemented these interfaces can be used as a resource inside "try with resource" syntax.

eg:: All java.io package classes,java.sql pacakge class has implemented these interfaces so we can use "try with resource" syntax on all these classes.

Till jdk1.6 compulsorily we need to have catch or finally assoicated with try block otherwise it is a compile time error.

a. try{

} //invalid

b. try{

}finally{

}

c. try{

}catch(XXXX e){

}

From JDK1.7, we can write try without catch/finally also.

eg:: try(){

}

As noticed above we can say finally block becomes dummy infront of try with resources from jdk1.7 onwards.

**Snippet::**

**======**

**1. public class HelloWorld{**

**public static void main(String... args){**

**if(args[0].equals("hello") ? false:true)**

**System.out.println("SUCCESS");**

**else**

**System.out.println("FAILURE");**

**}**

**}**

What is the output if the file is compiled and executed?

javac HelloWorld.java

java HelloWorld hello

Options

A. CE

B. SUCCESS

C. FAILURE

D. RunTimeException

**2.public class DemoApp {**

**static int count = 0;**

**int i = 0;**

**public void changeCount() {**

**while (i < 5) {**

**i++;**

**count++;**

**}**

**}**

**public static void main(String[] args) {**

**DemoApp demoApp1 = new DemoApp();**

**DemoApp demoApp2 = new DemoApp();**

**demoApp1.changeCount();**

**demoApp2.changeCount();**

**System.out.println(DemoApp.count + ":" + DemoApp.count);**

**}**

**}**

What is the output?

A. 10: 10

B. 5: 5

C. 5: 10

D. Compilation fails

**3.class Alpha {**

**int ns;**

**static int s;**

**Alpha(int ns) {**

**if (s < ns) {**

**s = ns;**

**this.ns = ns;**

**}**

**}**

**void doPrint() {**

**System.out.println("ns = " + ns + " s=" + s);**

**}**

**}**

**public class DemoApp {**

**public static void main(String[] args) {**

**Alpha a1 = new Alpha(50);**

**Alpha a2 = new Alpha(125);**

**Alpha a3 = new Alpha(100);**

**a1.doPrint();**

**a2.doPrint();**

**a3.doPrint();**

**}**

**}**

Options

A. ns =50 s=125

ns =125 s=125

ns =100 s=125

B. ns =50 s=125

ns =125 s=125

ns =0 s=125

C. ns =50 s=50

ns =125 s=125

ns =100 s=100

D. ns =50 s=50

ns =125 s=125

ns =0 s=125

**Exception Handling enchancment**

**=============================**

* try with resource
* try with multicatch

**try with resource(jdk1.7)**

**=========================**

=> With the help of try with resource, as a programmer i can just focus on

buisness logic rather than closing the resources.

=> Syntax::

try(R1;R2;R3;......){

//logic of using resources

}catch(XXXX e){

//handling logic

}

R1,R2,R3 => resources used inside try block.

We can use any no of resources when we work with try with resources syntax,

but every resource should be terminated with ; symbol.

R1,R3,R3,.... => all these resources should directly or indirectly implements

java.lang.AutoCloseable interface, only then we can use otherwise it would

result in "CE".

**TrywithMultiCatchBlock**

**======================**

Till jdk1.6 :: even though handling code is common for few exception as a programmer we need to write seperately in every catch block, as a result of which the code would be redudant and it is lengthy which disturbs the readablity and the performance, to resolve this problem in jdk1.7 oracle team has introduced as new feature called "trywithmulticatchblock".

eg#1.

**import java.io.\*;**

**public class TestApp {**

**public static void main(String[] args){**

**try{**

**//risky code**

**........**

**........**

**........**

**........**

**}**

**catch (ArithmeticException ae){**

**//handling code**

**ae.printStackTrace();**

**}catch(NullPointerException ne){**

**//handling code**

**ae.printStackTrace();**

**}**

**catch(ClassCastException ce){**

**//handling code**

**System.out.println(ce.getMessage());**

**}catch(IOException ie){**

**//handling code**

**System.out.println(ce.getMessage());**

**}**

**}**

**}**

As noticed above even though the handling code of ArithmeticException and NullPointerException is same still we wrote seperate catch block to write a hanlding which increases the lines of code and it distrubs the readability of the application,to resolve this problem we use "trywithmulticatch" block apporach from jdk1.7 onwards.

Syntax::

**try{**

**//risky code**

**}catch(E1 | E2 |E3 .... e){**

**//handling code**

**}**

E1,E2,E3, ... are all exception object and this feature is useful only when the hanlding code is common for all the types of exception occured, otherwise use try with catch /finally or try with resource.

To use MultiCatch block ,we need to keep exception object in such a way they should not be related either as parent,child or same type otherwise it would result in "CompileTimeError".

eg#1.

**public class TestApp {**

**public static void main(String[] args){**

**try{**

**//risky code**

**}**

**catch (ArithmeticException |NullPointerException e){**

**//handling code**

**e.printStackTrace();**

**}**

**catch(ClassCastException |IOException e){**

**//handling code**

**System.out.println(e.getMessage());**

**}**

**}**

**}**

Note::

**try{**

**//risky code**

**}catch (ArithmeticException |Exception e){**

**//handling code**

**e.printStackTrace();**

**}**

The above code wont' compile because ArithmeticException and Exception are related.

**Exception Propogation/ducking**

**============================**

Whenever a method gets called by the caller, in the called method if the exception occurs automatically that method will communicate with jvm and generates the Exception Object,this Exception object will be handed over to jvm ,jvm will check wheter in that method their is a handling code written or not, if handling code is not available then automatically that exception object will be Ducked

/propogated to the caller,This approach is called as "Excpetion Propogation".

**Rethrowing an Exception**

**=======================**

To convert one exception type to another excpetion type,we use rethrowing concept.

eg#1.

**public class TestApp {**

**public static void main(String[] args){**

**try{**

**System.out.println(10/0);**

**}catch (ArithmeticException ae){**

**throw new NullPointerException();**

**}**

**}**

**}**

Output::

Exception in thread "main" java.lang.NullPointerException

at TestApp.main(TestApp.java:8)

Note::

1. It is valid,Exception is partially checked, we can write a handling code to

handle the Exception so we normally use this syntax.

**try{**

**}catch(Exception e){**

**}**

2. It is valid, but Error can be caught but we can't write a handling code ,so we

dont' use catch Error Objects.

**try{**

**}catch(Error e){**

**}**

Difference b/w new vs newInstance()?

new :: It is an operator which is used to create an object of a class, where the name of the class for which the object has to be created is known at the Begining itself.

If the required .class file is not available at the run time for jvm to execute then

it would result in "NoClassDefFoundError", which is an uncheckedException.

To use new operator , a class need not contain any argument constructor also.

eg::

**class Test{**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**new Test();**

**}**

**}**

newInstance():: It is a method which is used to create an object, this approach is

preferedwhen we are not aware of the name of the class for which

an object should be created is known from begining, it is known

only during "runtime".

If the required .class file is not available at the runtime then it

would result in "ClassNotFoundException" which is a

checkedexception.

To use newInstance() compulsorily the class should contain zero

argument/default constructor otherwise it would result in

"InstantiationException".

if the constructor is private then also newInstance() would throw

an exception called "IllegalAccessException".

eg#1.

**class Test{**

**static{**

**System.out.println("Loading of .class file");**

**}**

**public Test(){**

**System.out.println("Constructor got called");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args)throws Exception{**

**Object o = Class.forName(args[0]).newInstance();**

**}**

**}**

Output::

Loading of .class file.

Constructor got called.

Difference b/w ClassNotFoundException and NoClassDefFoundError?

Difference b/w instanceof vs isInstance()?

In order to load the .class file explicitly through java program we need to use

public static Class forName(String className) throws ClassNotFoundException

forName is a static method available inside inbuilt class called "Class".

eg:: class.forName(args[0])

=> public Object newInstance()

=> This method is used to create an object of the loaded class.

=> return type of the method is Object.

eg#1

**class Test{**

**static{**

**System.out.println("loading of .class file");**

**}**

**Test(){**

**System.out.println("Zero arg constructor");**

**}**

**}**

**public class TestApp{**

**public static void main(String... args) throws ClassNotFoundException{**

**Object o=Class.forName(args[0]).newInstance();**

**System.out.println(o.getClass().getName());**

**}**

**}**

output::

javac TestApp.java

java TestApp Test

Loading of .class file

Zero arg constructor

Test

Difference b/w NoClassDefFoundError vs ClassNotFoundException?

NoClassDefFoundError :: It is an uncheckedException.

For the hard coded class if the required .class file is

not available at the runtime then it would result in

"NoClassDefFoundError".

Test t=new Test();

Test.class file not availabe at runtime then"NoClassDefFoundError".

ClassNotFoundException:: It is an checkedException.

For the dynamic class if the required .class file is not

available at the runtime then it would result in

"ClassNotFoundException".

Class.forName(args[0]).newInstance();

args[0] supplied class if it is not available then it would result in

"ClassNotFoundException".

Difference b/w instanceof vs isInstance() ?

Q.Identify which one is legal

a. instanceOf

b. instanceof

c. instanceOF

d. None of the above.

instanceof :: It is used to check whether the given object is of particular type or

not.

It returns a boolean value true if the type is of parenttype,same type

otherwise it returns false.

Syntax:: A instanceof B

A=> reference

B=> ClassName/Interface

instanceof operator is used when the type we want to check with is already

availabe from the begining itself.

eg#1.

**String s=new String("sachin");**

**System.out.println(s instanceof Object);**//true

eg#2.

**Object o=new String("sachin");**

**System.out.println(o instanceof Object);**//true

eg#3.

**Object o=new String("sachin");**

**System.out.println(o instanceof StringBuilder);**//false

eg#4.

**String s=new String("sachin");**

**System.out.println(s instanceof String);**

**System.out.println(s instanceof StringBuilder);** //CE:incompatible types

**System.out.println(s instanceof StringBuffer);** //CE :incompatible types

Note::

Object(C)

|

|======> Runnable(I)

|

Thread(C)

eg#5.

Thread t= new Thread();

System.out.println(t instanceof Object);//true

System.out.println(t instanceof Thread);//true

System.out.println(t instanceof Runnable);//true

eg#5.

Object o =new Object();

System.out.println(o instanceof Object);//true

System.out.println(o instanceof String);//false

System.out.println(o instanceof StringBuilder);//false

System.out.println(o instanceof Thread);//false

System.out.println(o instanceof Runnable);//false

eg#6.

System.out.println(null instanceof Object);

System.out.println(null instanceof String);

System.out.println(null instanceof StringBuilder);

System.out.println(null instanceof Thread);

Note:: null instanceof with any type the result is always "false".

null -> reserve word used to assign default value for all reference type.

eg:: Arrays,class,interface,enum.

true,false -> reserve word meant for boolean datatypes.

eg#7.

**interface Sample{**

**}**

**class Demo1 implements Sample{**

**}**

**class Demo2 implements Sample{**

**}**

**class Demo3{**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Sample s=new Demo1();**

**System.out.println(s instanceof Sample);//true**

**Demo3 d=new Demo3();**

**System.out.println(d instanceof Sample);//false**

**}**

**}**

**Usage of instanceof is used in "Collections".**

eg#8.

**interface Fish {}**

**class Perch implements Fish {}**

**class Walleye extends Perch {}**

**class Bluegill {}**

**public class Fisherman {**

**public static void main(String[] args) {**

**Fish f = new Walleye();**

**Walleye w = new Walleye();**

**Bluegill b = new Bluegill();**

**if (f instanceof Perch)**

**System.out.print("f-p ");**

**if (w instanceof Fish)**

**System.out.print("w-f ");**

**if (b instanceof Fish)**

**System.out.print("b-f ");**

**}**

**}**

What is the result?

A. w-f

B. f-p w-f

C. w-f b-f

D. f-p w-f b-f

E. Compilation fails.

F. An exception is thrown at runtime.

isInstance():: This method is used to check whether the object is of particular type

or not,but that type is not known from begining.

eg::

**class Test{**

**}**

**public class TestApp {**

**public static void main(String[] args)throws ClassNotFoundException{**

**Test t = new Test();**

**System.out.println(Class.forName(args[0]).isInstance(t));**

**}**

**}**

**MultiThreading**

**=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=**

a. Task/Process

Any program which is under execution is called as "Process"/"task".

The program under execution will always be under the control of "OS".

If "OS" is Mulitaskingos then it has the capable to load mulitple process on

the RAM and make sure the cpu time is effectively utilised.

b. Multitasking

Making sure the mulitple process are running smoothly under the control of

OS is refered as "Mulittasking".

eg:: While typing java program, we can listen to song,we can send a mail.

All these task are running smoothfully without any lag,means the operating

system we have is supporting "MultiTasking".

=> It refers to process of executing indepedent tasks simultaneously is refered

as "MulitTasking".

What is Thread?

What is the difference b/w task and Thread?

What is the difference b/w multitasking and mulitthreading?

How to create an thread?

In java how does multithreading is used?

How to create muliple threads for our application?

....

...

...

...

Backward link for StringBuilder(1.5v) vs StringBuffer(1.2V)

**Snippet::**

Given:

**1. public class Boxer1{**

**2. Integer i;**

**3. int x;**

**4. public Boxer1(int y) {**

**5. x = i+y;**

**6. System.out.println(x);**

**7. }**

**8. public static void main(String[] args) {**

**9. new Boxer1(new Integer(4));**

**10. }**

**11.}**

What is the result?

A. The value "4" is printed at the command line.

B. Compilation fails because of an error in line 5.

C. Compilation fails because of an error in line 9.

D. A NullPointerException occurs at runtime.

E. A NumberFormatException occurs at runtime.

F. An IllegalStateException occurs at runtime.

Given:

**public static void test(String str) {**

**int check = 4;**

**if (check = str.length()) {**

**System.out.print(str.charAt(check -= 1) +", ");**

**} else {**

**System.out.print(str.charAt(0) + ", ");**

**}**

**}**

**and the invocation:**

**test("four");**

**test("tee");**

**test("to");**

What is the result?

A. r, t, t,

B. r, e, o,

C. Compilation fails.

D. An exception is thrown at runtime.

What will be the result of compiling and executing Test class?

**public class Test {**

**public static void main(String[] args) {**

**char [][] arr = {**

**{'A', 'B', 'C'},**

**{'D', 'E', 'F'},**

**{'G', 'H', 'I'}**

**};**

**for(int i = 0; i < arr.length; i++) {**

**for(int j = 0; j < arr[i].length; j++) {**

**System.out.print(arr[i][1]);**

**}**

**System.out.println();**

**}**

**}**

**}**

A. ABC

DEF

GHI

B. BBB

EEE

HHH

C. AAA

DDD

GGG

D. CCC

FFF

III

Task/process

============

Any program which is under execution is called as "process".

Multitasking

============

Executing multiple process simultaneously where each process is independent of

each other, such type of process execution is termed as "Multitasking".

The control of Multitasking is not at the user level, it will always be under the

control of "Operating System".

eg:: Typing a java program

listeneing to music

sending an email

....

....

Thread concept

==============

=> Line of execution is technically termed as "Thread".

=> For every java program jvm will create a line of execution, through which it

starts the execution.

=> With single line of execution always the performance will be low.

=> To resolve the performance issue we create our own lines of execution

technically called "Mult-Threading".

What is MulitThreading?

Executing mulitiple task, where each taks is a seperate independent task of the

same program is refered as "MulitThreading" or "ThreadBased MultiTasking".

Each seperate task executed by the line of control is termed as "Thread".

Advantages of MultiThreading in java

====================================

=> Through MultiThreading, we can improve the response time of the

application.

=> MultiThreading is under the control of the programmer through which he

create a multiple thread for his application to execute.

=> When compared with other language like c++, creating a thread is very easy

in java because we focus on only 10% of the code for thread creation

remaining part of the code will be taken care by the API like Thread

,ThreadGroup,ThreadLocal,......

=> Applications related to MultiThreading

a. To implement servers

b. To implement webbased applications like gmail,facebook,bms,....

c. To implement animation gaming applications,.......

**MultiTasking vs MultiThreading**

**===============================**

MultiTasking => It is associated with O.s

Context switching is a costly process

Many process are involved so they share different address space.

MulitThreading => It is associated with programmer(language)

Scheduling a thread is not a costly process.

Mulitiple threads will share a same address space so performance is

high.

Different ways to create a thread manually by the programmer

=============================================

There are 2 ways

a. By extending Thread class

b. By implementing Runnable interface

Day:: 20/12/2021

Topic :MultiThreading-3

==================

**Snippet::**

**1.class TestException extends Exception {**

**public TestException() {**

**super();**

**}**

**public TestException(String s) {**

**super(s);**

**}**

**}**

**public class Test {**

**public void m1() throws \_\_\_\_\_\_\_\_\_\_ {**

**throw new TestException();**

**}**

**}**

For the above code, fill in the blank with one option.

A. Exception

B. Object

C. RunTimeException

D. Error

Answer:: A

**2.public class Test {**

**private static int [] arr;**

**public static void main(String [] args) {**

**if(arr.length > 0 && arr != null) {**

**System.out.println(arr[0]);**

**}**

**}**

**}**

Predict Output, if the above code is run with given command?

java Test

A. CompilationError

B. No Output is produced

C. NullPointerException is thrown at runtime

D. ArrayIndexOutOfBoundsException is thrown at runtime

Answer:: C

**3.import java.io.FileNotFoundException;**

**import java.io.IOException;**

**abstract class Super {**

**public abstract void m1() throws IOException;**

**}**

**class Sub extends Super {**

**@Override**

**public void m1() throws IOException {throw new FileNotFoundException();}**

**}**

**public class Test {**

**public static void main(String[] args) {**

**Super s = new Sub();**

**try {**

**s.m1();**

**} catch (IOException e) {**

**System.out.print("A");**

**} catch(FileNotFoundException e) {**

**System.out.print("B");**

**} finally {**

**System.out.print("C");**

**}**

**}**

**}**

A. AC

B. BC

C. class sub gives compilation error

D. class Test gives compilation error

Answer:D

**4.import java.io.FileNotFoundException;**

**import java.io.IOException;**

**abstract class Super {**

**public abstract void m1() throws IOException;**

**}**

**class Sub extends Super {**

**@Override**

**public void m1() throws IOException {throw new FileNotFoundException();}**

**}**

**public class Test {**

**public static void main(String[] args) {**

**Super s = new Sub();**

**try {**

**s.m1();**

**} catch (FileNotFoundException e) {**

**System.out.print("X");**

**} catch (IOException e) {**

**System.out.print("Y");**

**} finally {**

**System.out.print("Z");**

**}**

**}**

**}**

A. XZ

B. YZ

C. XYZ

D. compilationerror.

Answer:XZ

**4.import java.io.FileNotFoundException;**

**import java.io.IOException;**

**abstract class Super {**

**public abstract void m1() throws IOException;**

**}**

**class Sub extends Super {**

**@Override**

**public void m1() throws IOException {throw new FileNotFoundException();}**

**}**

**public class Test {**

**public static void main(String[] args) {**

**Super s = new Sub();**

**try {**

**s.m1();**

**} catch (FileNotFoundException e) {**

**System.out.print("M");**

**} finally {**

**System.out.print("N");**

**}**

**}**

**}**

A. MN

B. N

C. CompilationError

D. Program ends abruptly

**Different ways of creating a Thread**

**==========================**

1. By extending Thread(java.lang.\*)

2. By implementing Runnable(java.lang.\*)

**Creating a UserDefined Thread using ThreadClass**

**=\*==================================\*=**

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("CHILD THREAD");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("MAIN THREAD");**

**}**

**}**

**}**

Output::

MAIN THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

or

MAIN THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

case studies

===========

1. About ThreadScheduler

ThreadScheduler is a part of jvm, which is capable of scheduling the threads.

Internally ThreadScheduler will use few algorithms through which

scheduling of a thread would happen.

Programmer can't predict which thread which get a chance for Execution

,because the internal algorithm of ThreadScheduler would vary from vendor

to vendor implementation of JVM.

Due to which we can't predict exact output rather the possible cases of output

can only be predicted.

2. difference b/w t.start() and t.run()

a. t.start()

On a thread object if we call t.start(), internally Thread class start() will

be called by "JVM".

internal logic of Thread class start

public void start(){

1. Register the userdefined thread to ThreadScheduler.

2. perform all other mandatory low level activites

3. internally make a call to run()

}

b. t.run()

if we explicitly make a call to run(), then no new thread will be created

rather the run() method will be executed just like normal method, due to

which we can exactly predict the output.

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("CHILD THREAD");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.run();**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("MAIN THREAD");**

**}**

**}**

**}**

Output:: In this program only one thread will be created by JVM, and the same

thread will be used to call run().

CHILD THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

CHILD THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

Note:: start() will actually do many internal logics required for MultiThreading

so we say start() as heart of "MultiThreading".

Without the execution of start(), we cannot bring "MultiThreading" in java.

Case1:: Overriding start() in userdefined Thread class.

It is not a good practise to overrdide start() of Thread class, becoz we

would not be getting the benefit of inheritance here through which many

internally activities we need to do,if we are unable to do then we can't

bring in "MultiThreading" in our applications.

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void start(){**

**System.out.println("start() of Thread class");**

**}**

**@Override**

**public void run(){**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("CHILD THREAD");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("MAIN THREAD");**

**}**

**}**

**}**

Output:: one Thread by created by JVM

start() of Thread class

MAIN THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

MAIN THREAD

Case2:: If we don't Override run() of Thread class, it is not a good practise becoz

simply a Thread is created but no job would be assigned to the Thread.

**public class Thread{**

**@Override**

**public void run(){**

**//empty implementation**

**}**

**}**

eg#1.

**class MyThread extends Thread{}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("MAIN THREAD");**

**}**

**}**

**}**

Output:: 2 Threads are created,

MainThread

a. MAIN THREAD

...

...

UserDefinedThread

no job

**Note::** Overriding start() is not appreciated, where as Overriding run() is

appreciated.

Case3::

Overloading of run() method in userdefined Thread class.

It is also possible to overload run(), but Thread class start() will always call zero

arg run(), if other run() has to be called then we need to explcitly make a call to

Overloaded run().

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**System.out.println("zero arg run()");**

**}**

**//specialized method**

**public void run(int i){**

**System.out.println("one arg run()");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**System.out.println("MAIN THREAD");**

**}**

**}**

Output:: MAIN THREAD

zero arg run()

or

zero arg run()

MAIN THREAD

Day:: 22/12/2021

Topic :MultiThreading-4

==================

**Snippet::**

**6.What will be the result of compiling and executing Test class?**

**public class Test {**

**private static void m1() {**

**System.out.println(1/0);**

**}**

**public static void main(String[] args) {**

**try {**

**m1();**

**} finally {**

**System.out.println("A");**

**}**

**}**

**}**

A. A is printed on the console and program terminates normally

B. A is printed on the console,stacktrace is printed and then program ends normally

C. A is printed on the console,stacktrace is printed and then program ends abruptly

D. Compilation Error.

Answer:: C

**7.Which of the following keywords is used to manually throw an exception?**

**A. throw**

**B. thrown**

**C. throws**

**D. catch**

**8.What will be the result of compiling and executing Test class?**

**public class Test {**

**private static void m1() throws Exception {**

**throw new Exception();**

**}**

**public static void main(String[] args) {**

**try {**

**m1();**

**} finally {**

**System.out.println("A");**

**}**

**}**

**}**

A. A is printed on the console and program terminates normally

B. A is printed on the console,stacktrace is printed and then program ends normally

C. A is printed on the console,stacktrace is printed and then program ends abruptly

D. Compilation Error.

**9.Which of the following is a checked Exception?**

**A. ClassCastException**

**B. FileNotFoundException**

**C. ExceptionInIntializerError**

**D. RunTimeException**

ANSWER: B

**10.What will be the result of compiling and executing Test class?**

**public class Test {**

**private static String s;**

**public static void main(String[] args) {**

**try {**

**System.out.println(s.length());**

**} catch(NullPointerException | RuntimeException ex) {**

**System.out.println("DONE");**

**}**

**}**

**}**

A. DONE

B. Executes succesfully but no output

C. CompilationError

D. NoOutput

ANSWER: C

Thread

======

=> Overloading of start() is not appreciated, where as Overriding or run() is

appreciated.

=> start() is the only way in which we can promote MultiThreading in our

application.

=> start() is also called as "Heart of MultiThreading".

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void start(){**

**System.out.println("start() method invoked");**

**}**

**@Override**

**public void run(){**

**System.out.println("zero arg run()");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**System.out.println("MAIN THREAD");**

**}**

**}**

**Output:: Only one Thread (Main Thread)**

**start() method invoked**

**MAIN THREAD**

**eg#2.**

**class MyThread extends Thread{**

**@Override**

**public void start(){**

**super.start();**

**System.out.println("start() method invoked");**

**}**

**@Override**

**public void run(){**

**System.out.println("zero arg run()");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**System.out.println("MAIN THREAD");**

**}**

**}**

Output:: 2 Thread(One MainThread,One UserThread)

MainThread

start() method invoked

MAIN THREAD

UserDefinedThread

zero arg run()

Thread LifeCycle

================

new/born ============>ReadyState ====================>Running ==================> DeadState

t.start() T.S allocates time run() complete

* If thread is just created,we say thread is in "new/bornstate".
* If t.start() is invoked, we say the thread has entered into "ReadyState".
* If ThreadScheduler allocates cpu time for a thread, it means thread has entered into "RunningState".
* If run() is completed by a thread, then we say thread has entered into "DeadState".

**Note::** It is not possible to start the same thread again,if we try to do it would

result in "IllegalThreadStateException".

**class MyThread extends Thread{**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**;;;;;;;;;;;;;;;;;;**

**t.start();//JVM:: IllegalThreadStateException**

**}**

**}**

**Alternative way of creating a Thread using Runnable interface**

**=\*==========================================\*=**

**public interface Runnable{**

**public abstract void run();**

**}**

**public class Thread implements Runnable{**

**public void start(){**

**1. Register the thread with ThreadScheduler**

**2. perform low level mandatory activites**

**3. invoke run()**

**}**

**@Override**

**public void run(){**

**}**

**}**

eg#1.

**class MyRunnable implements Runnable{**

**@Override**

**public void run(){**

**for (int i=1; i<=10;i++ ){**

**System.out.println("Child thread");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyRunnable r=new MyRunnable();**

**Thread t= new Thread();**

**t.start();**

**for (int i=1; i<=10;i++ ){**

**System.out.println("Main thread");**

**}**

**}**

**}**

Output:: 2 Threads are created(MainThread,userDefinedThread)

MainThread

a. Main Thread

....

....

...

userDefinedThread

a. Empty(no job)

In the above code, JVM will execute Thread class run() due to which the userdefined line of execution would not perform anything.

eg#2.

**class MyRunnable implements Runnable{**

**@Override**

**public void run(){**

**for (int i=1; i<=10;i++ ){**

**System.out.println("Child thread");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyRunnable r=new MyRunnable();**

**Thread t= new Thread(r);//passing reference of MyRunnable made jvm to invoke**

**MyRunnable object run().**

**t.start();**

**for (int i=1; i<=10;i++ ){**

**System.out.println("Main thread");**

**}**

**}**

**}**

Output:: 2 Threads are created(MainThread,userDefinedThread)

MainThread

a. Main Thread

....

....

...

userDefinedThread

a. Child Thread

....

....

....

**Note::** Exact output we can't predict becoz to which Thread the ThreadScheduler will allocate a CPU time is not in the hands of programmer(based on algorithm).

**Snippet::**

**11.What will be the result of compiling and executing Test class?**

**class Base {**

**public void m1() throws NullPointerException {**

**System.out.println("Base: m1()");**

**}**

**}**

**class Derived extends Base {**

**public void m1() throws RuntimeException {**

**System.out.println("Derived: m1()");**

**}**

**}**

**public class Test {**

**public static void main(String[] args) {**

**Base obj = new Derived();**

**obj.m1();**

**}**

**}**

A. Base:m1()

B. Derived:m1()

C. CompilationError in Derived class

D. CompilationError in Base class

**12.Consider the following interface declaration:**

**public interface I1 {**

**void m1() throws java.io.IOException;**

**}**

**Which of the following incorrectly implements interface I1?**

A. public class c1 implements I1{

public void m1(){}

}

B. public class c1 implements I1{

public void m1()throws java.io.FileNotFoundException{}

}

C.public class c1 implements I1{

public void m1()throws java.io.IOException{}

}

D.public class c1 implements I1{

public void m1()throws Exception{}

}

**13.Which of the following are Java Exception classes? Select 3 options.**

A. ClassCastException

B. NullException

C. NumberFormatException

D. IllegalArgumentException

E. ArrayIndexException

**14.Given Code:**

**import java.io.\*;**

**class ReadTheFile {**

**static void print() { //Line 4**

**throw new IOException(); //Line 5**

**}**

**}**

**public class Test {**

**public static void main(String[] args) { //Line 10**

**ReadTheFile.print(); //Line 11**

**//Line 12**

**}**

**}**

**Which 2 changes are necessary so that code compiles successfully?**

A. replace Line 4 with static void print() throws Exception

B. replace Line 4 with static void print() throws Throwable

C. replace Line 10 with public static void main(String[] args) throws IOException

D. Surrond Line 11 with

try{

ReadTheFile.print();

}catch(IOException | Exception e){}

E. Surrond Line 11 with

try{

ReadTheFile.print();

}catch(Exception e){}

F. Surrond Line 11 with

try{

ReadTheFile.print();

}catch(IOException e){}

**15. public class Test{**

**public static void main(String... args){**

**args[1]="tendulkar";//line-n1**

**System.out.println(args[0]+":"+args[1]);//line-n2**

**}**

**}**

**javac Test.java**

**java Test sachin**

**What is the output?**

A. sachin tendulkar

B. CompilationError at line-n1

C. CompilationError at line-n2

D. ArrayIndexOutOfBoundsException at line-n1

E. ArrayIndexOutOfBoundsException at line-n2

**16.public class Test{**

**static String str="Good Job";**

**public static void main(String... args){**

**String str=str+" Mate!!!";//line-n1**

**System.out.println(str);//line-n2**

**}**

**}**

**What is the output?**

A. Good Job Mate!!!

B. Good Job

C. RunTimeException at line-n1

D. CompilationError at line-n1

E. CompilationError at line-n2

F. Mate!!!

Day:: 23/12/2021

Topic :MultiThreading-5

============================

**Thread Continuation**

**===================**

1. Extending Thread class

2. By implementing Runnable interface

**public inteface Runnable{**

**public abstract void run();**

**}**

**public class Thread implements Runnable{**

**public void start(){**

**1.register the thread with ThreadScheduler**

**2.perform low level mandatory activites**

**3.invoke run()**

**}**

**@Override**

**public void run(){**

**//Empty implementation**

**}**

**}**

Case Study

=========

MyRunnable r=new MyRunnable();

Thread t1=new Thread();

Thread t2 =new Thread(r);

case1:: t1.start();

upon invocation of t1.start(), userdefined thread will be created.

internally jvm will calls Thread class run() which has empty implementation.

Output::

MainThread

a. MainThread

....

....

....

UserDefinedThread

does nothing(empty)

case2:: t1.run()

upon invocation of t1.run(), no userdefined thread will be created.

jvm created mainThread, will call run() just like a normal method.

Thread class run() will be invoked which has emtpy implementation.

Output::

MainThread

a. calls run() which has empty implementation

b. MainThread

MainThread

......

case3:: t2.start()

upon invocation of t2.start(), a new userdefined thread will be created.

internally start() of Thread class will be called which internally invokes

run() of MyRunnable type.

Output(Depending on the ThreadScheduler respecitive thread will get a chance)

MainThread

a. MainThread

....

....

....

UserDefinedThread

a. ChildThread

....

....

case4:: t2.run()

Invocation of t2.run() would not create a new thread, rather jvm created

thread will execute run() just like a normal method.

run() of MyRunnable will be executed becoz Thread is created using

Thread t=new Thread(r);

Output::

MainThread

a. run() of MyRunnable class

Child Thread

Child Thread

.....

.....

b. MainThread

...

...

...

case5:: r.start()

It would result in compile time error, becoz w.r.t MyRunnable we don't

have start()

case6:: r.run()

Invocation of r.run() would not result in creation of new thread.

jvm created thred will just call run() like a normal method.

Output:(Only one Thread is created)

MainThread

ChildThread

......

......

MainThread

......

......

Question.

In which of the following case,a new Thread will be created which is responsible for executing MyRunnable run()?

a. t1.start()

b. t1.run()

c. t2.start() (only this option)

d. t2.run()

e. r.start()

f. r.run()

In which of the following case a new Thread will be created?

a. t1.start()

b. t1.run()

c. t2.start()

d. t2.run()

e. r.start()

f. r.run()

Answer:: option a, c

In which of the following case MyRunnable run() will be executed?

a. t1.start()

b. t1.run()

c. t2.start()

d. t2.run()

e. r.start()

f. r.run()

Answer:: option c,d,f

eg#1.

**class MyRunnable implements Runnable{**

**@Override**

**public void run(){**

**for (int i=1; i<=10;i++ ){**

**System.out.println("Child thread");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyRunnable r=new MyRunnable();**

**Thread t1=new Thread();**

**Thread t2= new Thread(r);**

**//stmt-1**

**for (int i=1; i<=10;i++ ){**

**System.out.println("Main thread");**

**}**

**}**

**}**

**replace stmt-1 with**

a. t1.start(),t1.run()

b. t2.start(),t2.run()

c. r.start(),r.run()

Which approach is best for defining a Thread in java?

Ans. There are 2 approaches

a. extending Thread class

b. implementing Runnable

* If the class is implementing Runnable approach to create a thread then that apporach is good becoz, the implementation class can not only implement but can also get the benefit of "inheritance".
* Memeory level performance is always good at interface level.
* if the class is extending the Thread class, then the same class cannot get the benefit of inheritance becoz in java mulitple inheritance is not supported through "class".
* Memeory level performance is not good at class level.

**Important Thread class Constructor**

**==================================**

Thread t= new Thread();

Thread t= new Thread(Runnable r);//commonly used to create a Thread object

Thread t= new Thread(String name);

Thread t= new Thread(Runnable r,String name);

Thread t= new Thread(ThreadGroup t, String name);

Thread t= new Thread(ThreadGroup t, Runnable r);

Thread t= new Thread(ThreadGroup t, Runnable r,String name);

Thread t= new Thread(ThreadGroup t, Runnable r,String name,long ms);

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**System.out.println("child thread");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t1=new MyThread();**

**Thread t2=new Thread(t1);**

**t2.start();**

**System.out.println("Main thread");**

**}**

**}**

Output::

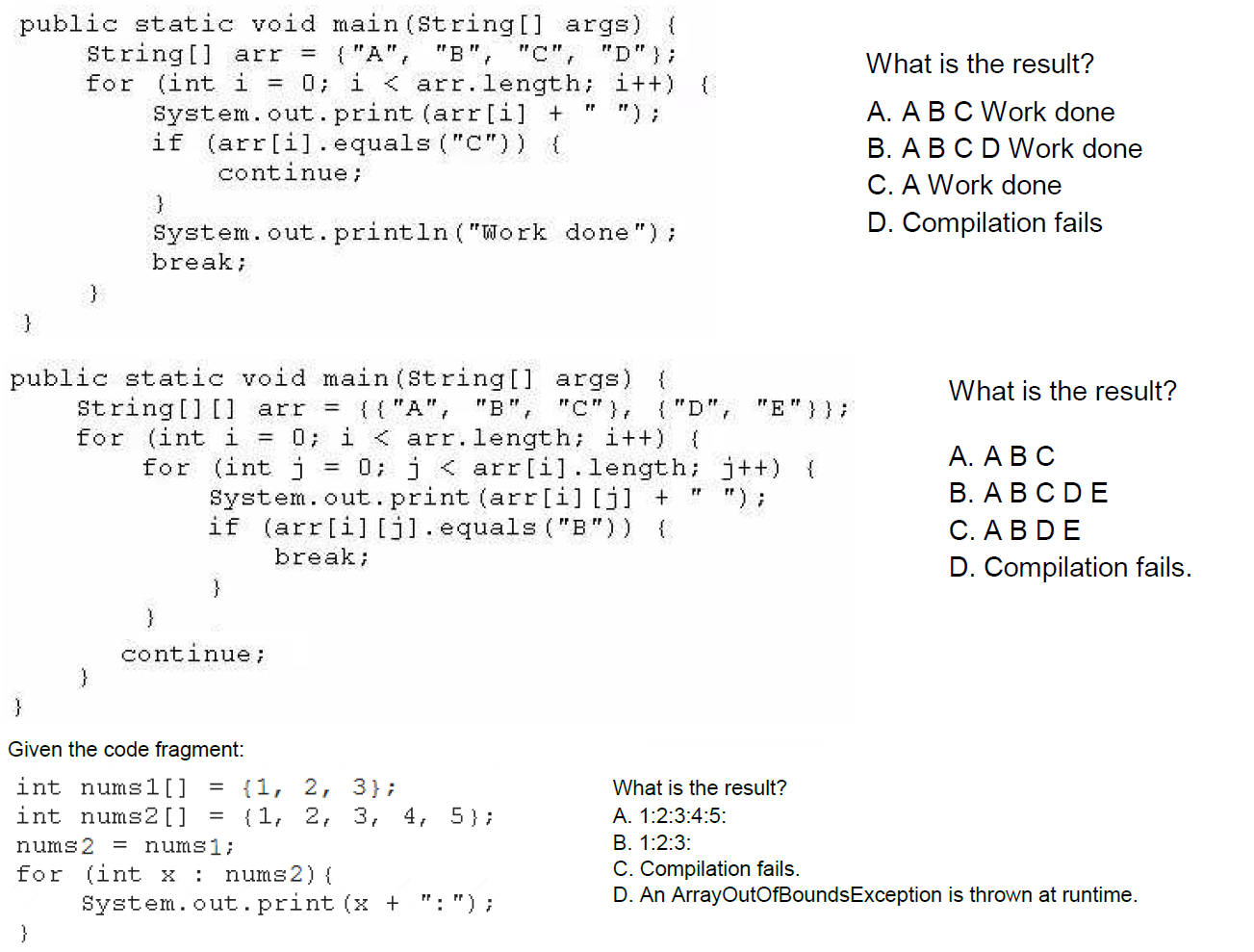
Main Thread

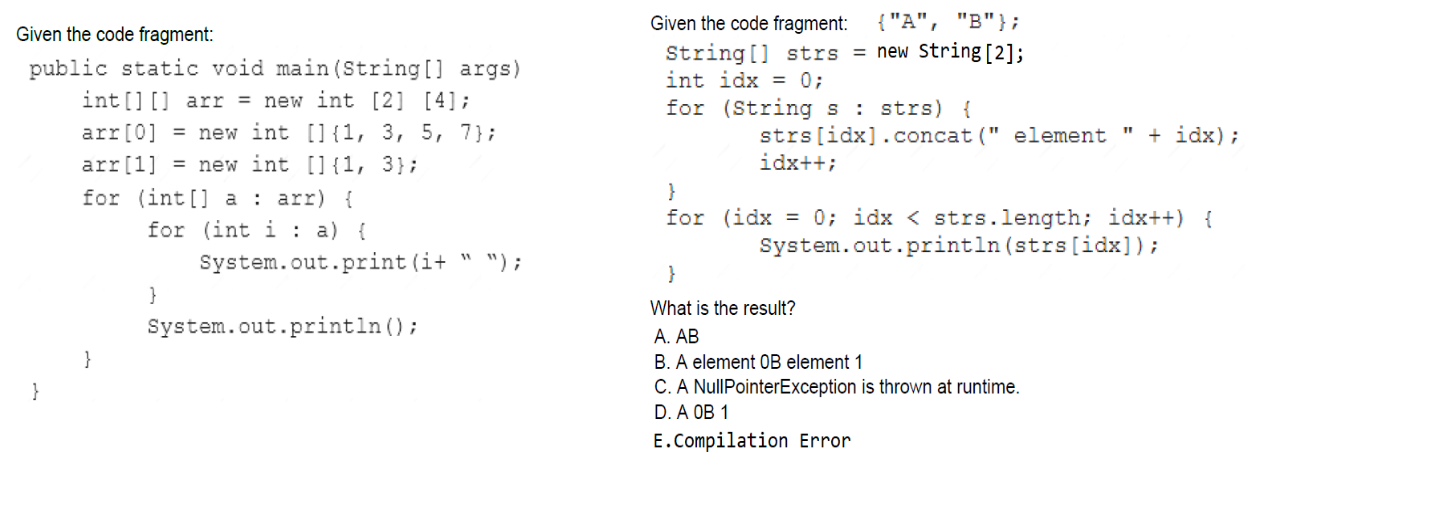
Child Thread

or

Child Thread

Main Thread

**Snippet::**



Day:: 24/12/2021

Topic :MultiThreading-6

============================

Different approach of Creating Thread

a. Runnable

b. Thread

public interface Runnable{

public abstract void run();

}

public class Thread implements Runnable{

//heart of MultiThreading

public void start(){

1. register the thread with Thread Scheduler

2. perform mandatory low level activites

3. invoke run() based on the Thread object creation.

a. run() => if Thread is created using new Thread();

b. run() of Runnable implementation object=> if Thread is created

using new Thread(Runnable r);

}

@Overriding

public void run(){

//empty implementation

}

}

**Name of the Thread**

**==================**

For every Thread in java, there would be name which would implicitly given

by jvm or explicitly given by the programmer.

It is possible to set the name of the the thread explicitly as well as it is possible

to get the name of the thread explicitly using "Thread class API".

public final void setName(String name)

public final String getName()

It is possible to get the Currently executing thread object reference using a method called Thread.currentThread()

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**Thread.currentThread().setName("Modi");//setting name for child thread**

**System.out.println("Child Thread name is :: "+Thread.currentThread().getName());**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**System.out.println("Main Thread name is :: "+Thread.currentThread().getName());**

**MyThread t1 =new MyThread();**

**t1.start();**

**Thread.currentThread().setName("yash");//setting the name of the thread**

**System.out.println("Thread under execution is ::"+Thread.currentThread().getName());**

**System.out.println(10/0);**

**}**

**}**

Output::

Main Thread name is :: main

Thread under execution is ::yash

Child Thread name is :: Modi

Exception in thread "yash" java.lang.ArithmeticException: / by zero

at TestApp.main(TestApp.java:19)

**Thread Priority**

**==============**

By default in java language for every thread their would be a priority.

Default priority of the thread is 5.

In Thread class we have 3 variables which would speak about min,max,normal

priority.

public static final int MIN\_PRIORITY=1;

public static final int NORM\_PRIORITY=5;

public static final int MAX\_PRIORITY=10;

ThreadScheduler will allocate cpu time for a thread based on "Thread priority

only".

If both the threads have same priority then predicting the output is not possible

becoz it is the duty of ThreadScheduler to schedule the Thread.

**Note::** When ever we change the priority of a thread, then by default the same

priority will be assigned to the child thread also.

eg#1.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**Thread.currentThread().setName("dhoni");**

**System.out.println("The currently executed "+Thread.currentThread().getName()+**

**" thread priority is :: "+Thread.currentThrea().getPriority());**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Thread.currentThread().setPriority(9);**

**MyThread t=new MyThread();**

**t.start();**

**Thread.currentThread().setName("yash");**

**System.out.println("The currently executed "+Thread.currentThread().getName()+**

**"thread priority is :: "+Thread.currentThread().getPriority());**

**}**

**}**

Output

The currently executed yash thread priority is :: 9

The currently executed dhoni thread priority is :: 9

eg#2.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**Thread.currentThread().setName("dhoni");**

**System.out.println("The currently executed "+Thread.currentThread().getName()+**

**" thread priority is :: "+Thread.currentThread().getPriority());**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.setPriority(9);//setting the priority only for userdefined thread(yash)**

**t.start();**

**Thread.currentThread().setName("yash");**

**System.out.println("The currently executed "+Thread.currentThread().getName()+**

**" thread priority is :: "+Thread.currentThread().getPriority());**

**}**

**}**

Output::

2 threads are created

a. mainthread(yash=> 5)

b. userdefinedthread(dhoni=> 9)

Thread Scheduler will schedule the Thread based on priority.

=> less value we assign more will be the priority, more value we assign less will

be the priority.

eg#3.

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for(int i=1;i<=5;i++)**

**System.out.println("child thread");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.setPriority(9);**

**t.start();**

**for(int i=1;i<=5;i++)**

**System.out.println("main thread");**

**}**

**}**

**Note::** As noticed above 2 threads are created,since the priority is set we can predict the exact output.since priority is set we can say a child thread should wait till the completion of main thread as a result of which the peformance would be low.

MainThread(priority => 5)

main thread

......

......

......

ChildThread(priority => 9)

child thread

......

......

......

**Note::** Priority of any Thread should always lie with in the range of 1-10.

if we try to give more than 10 or less than 1 then api would reply with an

exception called "IllegalArgumentException".

Day:: 27/12/2021

Topic :MultiThreading-6

============================

Different approach of Creating Thread

a. Runnable

b. Thread

public interface Runnable{

public abstract void run();

}

public class Thread implements Runnable{

//heart of MultiThreading

public void start(){

1. register the thread with Thread Scheduler

2. perform mandatory low level activites

3. invoke run() based on the Thread object creation.

a. run() => if Thread is created using new Thread();

b. run() of Runnable implementation object=> if Thread is created

using new Thread(Runnable r);

}

@Overriding

public void run(){

//empty implementation

}

}

**Priority of a Thread**

**====================**

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for(int i=1;i<=5;i++)**

**System.out.println("child thread");**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.setPriority(9);//priority is 9(high Priority)**

**t.start();**

**for(int i=1;i<=5;i++)**

**System.out.println("main thread");**

**}**

**}**

Output::

2 Threads

1. MainThread=> priority(5)

2. ChildThread=>priority(9)

Since child thread has more priority, Thread Scheduler will give its time for Child thread so

child thread(5 times)

main thread( 5 times)

To get this type of proper output,Operating System support should be available.

Operating system will get that extra support with the help of batch files.

**Methods available to stop the execution of Thread**

**=====================================**

1.yield()

2.join()

3.sleep()

**yield()**

**========**

=> It makes the currently executing thread to pause and give a chance for other

waiting threads with the same priority.

=> If all the waiting threads have low priority then the currently executing thread

only will continue its execution, remaining threads should wait.

=> If there are several thread with the same priority then we can't predict which

thread will get a chance for execution, it will be decided by the.Threadscheduler.

=> The thread which called yield method,when will it get a chance back also

can't be decided by the programmer it would be based on the mercy of

ThreadScheduler.

=> public static native void yield();

native means the implementation of the method is not given by java, it is

given in some other language.

**eg#1.**

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for(int i=1;i<=5;i++){**

**Thread.yield();**

**System.out.println("child thread");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.start();**

**for(int i=1;i<=5;i++)**

**System.out.println("main thread");**

**}**

**}**

Output::

Since Child thread calls yield method it would pause its execution, so other thread with the same priority will get a chance and it continue with its execution.

2 Threads

a. Main Thread(priority = 5)

main thread

.....

.....

b. Child Thread(priority = 5)

child thread

child thread

**Note::** Few operating system might not support yield() call.

**join()**

**======**

if one thread wants to wait,till the other thread finsihes the execution then that particular thread which needs to wait should call a method called "join".

eg:: if t2 thread wants to wait, till t1 thread finsihes the execution then t2 thread should invoke a method called "t2.join()".

In the above case t2 thread will enter into waiting state, till t1 finishes its

execution.

Whenever the thread is in waiting state, if another thread interrupts then it would result in a RuntimeException called "InterruptedException".

**method prototype of join()**

**===================**

public final void join() throws InterruptedException;

public final void join(long millseconds) throws InterruptedException;

public final void join(long milliseconds, int ns) throws InterruptedException;

=> join() is final and it is overloaded method.

=> method is final,means it wont pariticpate in Overriding.

=> It is a instance method which can be called only on object.

**eg#1.**

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for(int i=1;i<=10;i++){**

**System.out.println("rama thread");**

**try{**

**Thread.sleep(2000);//2sec he is not doing anything**

**}catch(InterruptedException e){}**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**MyThread t=new MyThread();**

**t.start();**

**t.join();**//making the main thread to wait till child thread finishes its

execution.

**for(int i=1;i<=10;i++)**

**System.out.println("sita thread");**

**}**

**}**

Output::

2Threads

a. Main thread(priority-5)

printing rama ten times and making sita to wait till printing is completed.

b. child thread(priority-5)

print sita ten times

**eg#2.**

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for(int i=1;i<=10;i++){**

**System.out.println("rama thread");**

**try{**

**Thread.sleep(2000);**//2sec he is not doing anything

**}catch(InterruptedException e){}**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**MyThread t=new MyThread();**

**t.start();**

**t.join(10000**);//making the main thread(sita thread) waiting only for 10 sec,

if the execution is not completed then sita thread will continue

with its execution.

**for(int i=1;i<=10;i++)**

**System.out.println("sita thread");**

**}**

**}**

Output::

2Threads

a. Main thread(priority-5)

printing rama for 10 seconds after that control went back to main thread

b. child thread(priority-5)

print sita ten times now give the control to rama thread for execution.

**Snippet::**

**1.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**for(int x = 10, y = 11, z = 12; y > x && z > y; y++, z -= 2) {**

**System.out.println(x + y + z);**

**}**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. 33

B. 32

C. 34

D. 35

E. CompilationError

**2.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**int i = 0;**

**for(System.out.print(i++); i < 2; System.out.print(i++)) {**

**System.out.print(i);**

**}**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. 112

B. 012

C. 011

D. 12

E. 01

F. CompilationError

**3.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**boolean flag = false;**

**do {**

**if(flag = !flag) { //Line n1**

**System.out.print(1); //Line n2**

**continue; //Line n3**

**}**

**System.out.print(2); //Line n4**

**} while(flag); //Line n5**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. 1

B. 2

C. 12

D. 21

E. 212

F. 121

G. 112

H. 221

I. CompilationError

**4.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**String arr1 [], arr2, arr3 = null; //Line n1**

**arr1 = new String[2];**

**arr1[0] = "A";**

**arr1[1] = "B";**

**arr2 = arr3 = arr1; //Line n2**

**log(arr2); //Line n3**

**}**

**private static void log(String... vals) {**

**for(String s : vals)**

**System.out.print(s);**

**}**

**}**

A. Line n1 causes compile time error

B. Line n2 causes compile time error

C. Line n3 causes compile time error

D. Prints AB on the console

E. Prints BA on the console

F. Prints A on the console

G. Prints B on the console

**5.Given**

**public class A {**

**2. public String doit(int x, int y){**

**3. return "a";**

**4. }**

**5.**

**6. public String doit(int... vals){**

**7. return "b";**

**8. }**

**9. }**

**Given:**

**25. A a = new A();**

**26. System.out.println(a.doit(4, 5));**

**What is the result?**

A. Line 26 prints "a" to System.out.

B. Line 26 prints "b" to System.out.

C. An exception is thrown at line 26 at runtime.

D. Compilation of class A will fail due to an error in line 6.

**6.Given**

**class Alien {**

**String invade(short ships) { return "a few"; }**

**String invade(short... ships) { return "many"; }**

**}**

**class Defender {**

**public static void main(String [] args) {**

**System.out.println(new Alien().invade(7));**

**} }**

**What is the result?**

A. many

B. a few

C. Compilation fails

D. The output is not predictable

E. An exception is thrown at runtime

**7.Given code:**

**public class Test {**

**public static void main(String[] args) {**

**String [] arr = {"I", "N", "S", "E", "R", "T"};**

**for(/\*INSERT\*/) {**

**if (n % 2 == 0) {**

**continue;**

**}**

**System.out.print(arr[n]); //Line n1**

**}**

**}**

**}**

**And below options:**

**1. int n = 0; n < arr.length; n += 1**

**2. int n = 0; n <= arr.length; n += 1**

**3. int n = 1; n < arr.length; n += 2**

**4. int n = 1; n <= arr.length; n += 2**

**How many above options can be used to replace /\*INSERT\*/, such that on execution,**

**code will print NET on to the console?**

A. only one option

B. only two option

C. only three option

D. all 4 options

E. None of the options

Date: 28/12/2021

Topic: MultiThreading-6

====================

**MultiThreading**

**===============**

Making child thread to wait till the main thread finsihes the execution.

**eg#1.**

**class MyThread extends Thread{**

**static Thread mt;**

**@Override**

**public void run(){**

**try{**

**mt.join();**

**}catch(InterruptedException ie){}**

**for (int i=1;i<=10 ;i++ ){**

**System.out.println("child thread");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**MyThread.mt=Thread.currentThread();**

**MyThread t1=new MyThread();**

**t1.start();**

**for (int i=1;i<=10 ;i++ ){**

**System.out.println("main thread");**

**Thread.sleep(2000);**

**}**

**}**

**}**

Output

2 Threads

a. Main Thread(priority=5)

will print main thread 10 times only then child thread will get a chance

b. Child Thread(priority=5)

will print child thread 10 times

**Note::**

if child thread waits for main thread and if main thread wait for child thread then both these threads would wait forever then the program would be hanged

(technically it called as DeadLock).

if currently executing thread invokes join method on itself then also it would result in "DeadLock".

**eg#1.**

**class MyThread extends Thread{**

**static Thread mt;**

**@Override**

**public void run(){**

**try{**

**mt.join();//child thread waiting for main thread**

**}catch(InterruptedException ie){}**

**for (int i=1;i<=10 ;i++ ){**

**System.out.println("child thread");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**MyThread.mt=Thread.currentThread();**

**MyThread t1=new MyThread();**

**t1.start();**

**t1.join();//main thread waiting for child**

**for (int i=1;i<=10 ;i++ ){**

**System.out.println("main thread");**

**Thread.sleep(2000);//2sec**

**}**

**}**

**}**

Output

2 Threads

a. Main Thread

main thread waiting for child thread so no processing

b. Child Thread

child thread waiting for main thread so no processing

**eg#2.**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**Thread.currentThread().join();**

**}**

**}**

Output

One Thread

A. main

it wait for himself so it result in dead lock.

**sleep()**

**======**

If a thread does not want to perform any operation for a particular period of time then we need to go for sleep method.

method prototype of sleep method

=========================

public static native void sleep(long) throws java.lang.InterruptedException;

public static void sleep(long, int) throws java.lang.InterruptedException;

sleep method call would result in checked Exception called InterruptedException so compulsorily it should be handled through try catch or using throws.

**eg#1.**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**for (int i=1;i<=5 ;i++ ){**

**System.out.println("slide: "+i);**

**Thread.sleep(3000);//2sec**

**}**

**}**

**}**

**Interruption**

**============**

How a thread can interrupt another thread?

If a thread is in waiting state/ sleeping state we can interrupt a thread.

**Snippet::**

**1.Consider the following class**

**public class Test {**

**public static void main(String[] args) {**

**boolean b1 = 0;**

**boolean b2 = 1;**

**System.out.println(b1 + b2);**

**}**

**}**

A. 0

B. 1

C. true

D. false

E. CompilationError

**2.Consider below code of Wall.java file:**

**public class Wall {**

**public static void main(String args[]) {**

**double area = 5.7;**

**String color;//line-n1**

**if (area < 7)**

**color = "BLUE";//line-n2**

**System.out.println(color);//line-n3**

**}**

**}**

**What will be the result of compiling and executing Wall class?**

A. BLUE

B. null

C. Exception is thrown at runtime

D. CompilationError at line-n1

E. CompilationError at line-n2

F. CompilationError at line-n3

**3.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**double price = 90000;**

**String model;**

**if(price > 100000) {**

**model = "Tesla Model X";**

**} else if(price <= 100000) {**

**model = "Tesla Model S";**

**}**

**System.out.println(model);**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. Tesla Model X

B. Tesla Model S

C. null

D. CompilationError

**4.Given code of Test.java file:**

**public class Test {**

**static String msg; //Line 2**

**public static void main(String[] args) {**

**String msg; //Line 4**

**if(args.length > 0) {**

**msg = args[0]; //Line 6**

**}**

**System.out.println(msg); //Line 8**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. null

B. Line 2 causes compilation Error

C. Line 4 causes compilation Error

D. An Exception is thrown at Runtime

E. Line 8 causes compilation Error

**5.public class Test {**

**public static void main(String[] args) {**

**String s1 = "ABC";**

**String s2 = "abc";**

**System.out.println(/\*INSERT\*/);**

**}**

**}**

**Which of the following options, if used to replace /\*INSERT\*/, will compile successfully and on execution will print true on to the console?**

**Select 2 options.**

A. s1.equals(s2)

B. s1.equalsIgnoreCase(s2)

C. s2.equals(s2.toLower());

D. s1.equals(s2.toUpper());

E. s1.length() == s2.length()

F. s1.contentEquals(s2)

**6.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**String fName = "Joshua";**

**String lName = "Bloch";**

**System.out.println(fName = lName);**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. CompilationError

B. false

C. true

D. None of the above

**7.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**String str = "Think"; //Line n3**

**change(str); //Line n4**

**System.out.println(str); //Line n5**

**}**

**private static void change(String s) {**

**s.concat("\_Twice"); //Line n9**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. Think

B. \_Twice

C. Think\_Twice

D. None of the above

**8.Consider below code of Test.java file:**

**public class Test {**

**public static void main(String[] args) {**

**StringBuilder str = new StringBuilder("Think"); //Line n3**

**change(str); //Line n4**

**System.out.println(str); //Line n5**

**}**

**private static void change(StringBuilder sb) {**

**sb.append("\_Twice"); //Line n9**

**}**

**}**

**What will be the result of compiling and executing Test class?**

A. Think

B. \_Twice

C. Think\_Twice

D. None of the above

**9.class main {**

**static string main = "main";**

**public main(){**

**System.out.println("hiee");**

**}**

**public static void main(){**

**System.out.println("byee");**

**}**

**public static void main(String[] args) {**

**System.out.println("hello");**

**main();**

**}**

**}**

**What is the output?**

A.hiee

byee

hello

B. hello

hiee

byee

C. hello

byee

D. compilation fails

**10.System.out.println("1"+"2"+"3"+"4" == "1"+"2"+"3"+"4");**

A. Compilation fails

B. RunTimeException

C. Can't predict the output

D. 0

E. 1

F. 1234

G. None of the above

Date: 29/12/2021

Topic: MultiThreading-7

====================

**Thread continuation**

**===================**

If a thread is in waiting state/sleeping state/ just executing we can interrupt a thread by another thread using a method call called "interrupt()".

**prototype:**

public void interrupt();

**eg#1.**

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**try{**

**for (int i=1;i<=5; i++){**

**System.out.println("I am lazy Thread: "+i);//doing job**

**Thread.sleep(2000);//2 sec**

**}**

**}**

**catch (InterruptedException ie){**

**System.out.println("I got interrupted");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**MyThread t1=new MyThread();**

**t1.start();**

**t1.interrupt();//main thread is interrupting the child thread**

**System.out.println("end of main thread");**

**}**

**}**

Output

2 Threads

a. main(priority=5)

End of main thread

b. Thread-0(priority=5)

I am lazy thread : 1

I got interrupted

**eg#2.**

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for (int i=1;i<=5; i++){**

**System.out.println("I am lazy Thread: "+i);//doing job**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**MyThread t1=new MyThread();**

**t1.start();**

**t1.interrupt();//main thread is interrupting the child thread**

**System.out.println("end of main thread");**

**}**

**}**

Output

2 Threads

a. main(priority=5)

End of main thread

b. Thread-0(priority=5)

I am lazy thread : 1

I am lazy thread : 2

I am lazy thread : 3

I am lazy thread : 4

I am lazy thread : 5

**eg#3.**

**class MyThread extends Thread{**

**@Override**

**public void run(){**

**for(int i=1;i<=10000;i++)**

**System.out.println("Printing:: "+i);**

**System.out.println(" I am entering into sleeping state");**

**try{**

**Thread.sleep(2000);**

**}**

**catch (InterruptedException ie){**

**System.out.println("I got interrupted");**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args) throws InterruptedException{**

**MyThread t1=new MyThread();**

**t1.start();**

**t1.interrupt();//main thread is interrupting the child thread**

**System.out.println("end of main thread");**

**}**

**}**

Output

2 Threads

A. main

end of main thread

waiting for child thread for entering into sleeping/waiting state

B. Thread-0

printing 10000 data

I got interrupted.

**Note::**

1. Even if the main thread finishes its job interrupt() call waited for the child

threadto enter into sleeping state and if the child thread enters into sleeping state then it interrupted.

1. Even if the main thread finsihsed its job interrupt() call waited for child

thread to enter into sleeping state/waiting state.

if the child thread doesnot enter into sleeping/waiting then interrupt()

method call would be wasted.

1. Even if the main thread finsihes its job interrupt() call will wait for the child

thread to undergo sleeping/waiting till then interrupt() call would wait. once the child thread enters into sleeping/waiting state immediately the main thread would interrupt.

**Comparison b/w yield(),join() and sleep()**

**yield::**

**=======**

* + To pause the execution of current thread and give a chance for other

waiting threads who are of same priority.

* + It is static and native.
  + It is not final.
  + It is not overloaded method.
  + This method doesnot throw any checkedException.
  + If thread encounter yield it would move to ready/runnable state so
  + interrupt() usage is of no use.

**join::**

**=====**

* It makes a particular thread to wait for someother thread.
* It is not static and native.
* It is final.
* It is overloaded method.
* This method throws InterruptedException(Checked).
* If thread encounter join it would move to waiting state so interrupt() can

throw InterruptedException.

**sleep::**

**======**

* For a particular duration of time if a thread does not want to perform any

job then we use sleep.

* It is static and native sleep(long) and one more method is only static

sleep(long,int).

* It is overloaded method.
* This method throws InterruptedException(Checked).
* If thread encounter join it would move to sleeping state so interrupt() can

throw InterruptedException.

**InterviewQuestion**

**=================**

**class MyThread implements Runnable{**

**public static int i=3;**

**@Override**

**public void run(){**

**System.out.print(i--);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Thread[] arr=new Thread[3];**

**arr[0] =new Thread(new MyThread());**

**arr[1] =new Thread(new MyThread());**

**arr[2] =new Thread(new MyThread());**

**for(Thread t:arr)**

**t.start();**

**}**

**}**

Ans:: 321 output would be produced but can't predict the order of

Date: 03/01/2022

Topic: MultiThreading-8

===================

**Synchronization ::**

**================**

When multiple thread tries to access the same resource in multithreading style

then it would lead to "DataInconsitencyproblem".

To resolve this problem we need to learn about "Synchronization".

1. synchronized is a access modifier applicable at "method" level and "block"

level.

1. method or block declared as "synchronized", then at a time only one thread

can access.

synchronized access modifier deals with an "lock" for the object.

1. The main advantage of synchronized access modifier is to resolve

"DataInconsistencyProblem".

1. The main disadvantage of synchronized is it increases the waiting time for

the other threads becoz other threads need lock of the object,due to which the performance would be low,so we should not use synchronized whereever we want.

5. If a thread wants to execute any synchronized method on a particular

object,first the thread should get a lock of that object,using the lock the

thread will execute a particular method on that object, and after execution

it immediately releases the lock which it has acquired.

6. While a thread is executing any sycnhronized method,then remaining

threads are not allowed to execute any synchronized method on that object

simultaneously,But remaining threads can are allowed to execute non-

synchronized method on the same object, becoz lock is applied only for

synchronized methods not for non-synchronized methods.

7. "Lock concept is applied at Object level not at method level".

**eg#1.**

**class Display{**

**public synchronized void wish(String name){**

**for (int i=1;i<=10 ;i++ ){**

**System.out.print("Good Evening: ");**

**try{**

**Thread.sleep(2000);//2sec**

**}**

**catch (InterruptedException e){**

**e.printStackTrace();**

**}**

**System.out.println(name);**

**}**

**}**

**}**

**class MyThread extends Thread{**

**Display d;**

**String name;**

**MyThread(Display d,String name){**

**this.d=d;**

**this.name=name;**

**}**

**@Override**

**public void run(){**

**d.wish(name);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Display d= new Display();**

**MyThread t1= new MyThread(d,"dhoni");**

**MyThread t2= new MyThread(d,"yuvi");**

**t1.start();**

**t2.start();**

**}**

**}**

Output

3 threads

a. main thread

b. Thread-0

print dhoni 10 times(by acquiring lock)

c. Thread-1

print yuvi 10 times(after getting lock)

Date: 04/01/2022

Topic: MultiThreading-9

=======================

**Case study::**

Display d1= new Display();

Display d2= new Display();

MyThread t1= new MyThread(d1,"dhoni");

MyThread t2= new MyThread(d2,"yuvi");

Eventhough we declared wish() under synchronized,still we get irregular output in this case, because both the threads are operating on seperate object.

**Conclusion::**

If multiple threads are operating on mulitple object there there is no impact of

synchronization.

If mulitple threads are operation on single object then we need to use

synchronized keyword to avoid "datainconsistencyproblem".

**eg#1.**

**class Display{**

**public synchronized void wish(String name){**

**for (int i=1;i<=10 ;i++ ){**

**System.out.print("Good Evening: ");**

**try{**

**Thread.sleep(2000);//2sec**

**}**

**catch (InterruptedException e){**

**e.printStackTrace();**

**}**

**System.out.println(name);**

**}**

**}**

**}**

**class MyThread extends Thread{**

**Display d;**

**String name;**

**MyThread(Display d,String name){**

**this.d=d;**

**this.name=name;**

**}**

**@Override**

**public void run(){**

**d.wish(name);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Display d1= new Display();**

**Display d2= new Display();**

**MyThread t1= new MyThread(d1,"dhoni");**

**MyThread t2= new MyThread(d2,"yuvi");**

**t1.start();**

**t2.start();**

**}**

**}**

Output::

3 Threads

a. main

b. Thread-0

got lock of d1 and executing dhoni

c. Thread-1

got lock of d2 and executing yuvi

Nature:: irregular output

**eg#2.**

**class Display{**

**public synchronized void displayNumber(){**

**for (int i=1;i<=10 ;i++ ){**

**System.out.print(i);**

**try{**

**Thread.sleep(2000);//2sec**

**}**

**catch (InterruptedException e){**

**e.printStackTrace();**

**}**

**}**

**}**

**public synchronized void displayCharacter(){**

**for (int i=65;i<=75 ;i++ ){**

**System.out.print((char)i);**

**try{**

**Thread.sleep(2000);//2sec**

**}**

**catch (InterruptedException e){**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**class MyThread1 extends Thread{**

**Display d;**

**MyThread1(Display d){**

**this.d=d;**

**}**

**@Override**

**public void run(){**

**d.displayNumber();**

**}**

**}**

**class MyThread2 extends Thread{**

**Display d;**

**MyThread2(Display d){**

**this.d=d;**

**}**

**@Override**

**public void run(){**

**d.displayCharacter();**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Display d1= new Display();**

**MyThread1 t1= new MyThread1(d1);**

**MyThread2 t2= new MyThread2(d1);**

**t1.start();**

**t2.start();**

**}**

**}**

Output:: if we remove synchronized keyword then the output is irregular

3 Threads

a. main(priority-5)

b. Thread-0(priority-5)

c. Thread-1(priority-5)

A1B2C3D45EF6G78HI9J10K

Based on Threadscheduler, it will schedule the thread so output is "irregular".

**If we add synchronized keyword, then output is "regular".**

3 Threads

a. main(priority-5)

b. Thread-0(priority-5)

if Thread-0 gets a chance he applies lock on the object and till he

finsihes lock wont be released.

c. Thread-1(priority-5)

The thread will be in waiting state, because to execute the task he need

the lock of the display object.

ABCDEFGHIJK12345678910

**Classlevel lock**

**===============**

=> If a thread wants to call static synchronized method of class, then it needs

class level lock.

=> Class level lock are always unique.

=> if a thread gets a class level lock, then other threads to execute static

synchronized method of that class, it should wait.

=> if a thread gets object level lock,then other threads to execute synchronized

method of that object,it should wait.

=> To execute normal methods by a thread,lock concpet is not applicable.

=> lock concept is applicable at 2 levels

a. object level.

b. class level.

**eg:: class X{**

**static void m1(){}**

**synchronized void m2(){}**

**static synchronized void m3(){}**

**synchronized void m4(){}**

**static synchronized void m5(){}**

**void m6(){}**

**}**

t1=> m1() => lock is not applied

t2=> m2() => object level lock applied so other threads to execute synchronized

method of X object should wait.

t3=> m6() => lock is not applied

t4=> m3() => class level lock applied so other threads to execute static

synchronized method of X class should wait.

t5=> m5() => should wait becoz class level lock is with t4 thread.

Date: 05/01/2022

Topic: MultiThreading-10

========================

**synchronized block**

**public void m1(){**

**\*\*\*\*\*\***

**\*\*\*\*\*\***

**\*\*\*\*\*\***

**......**

**......//This region would make "DataInconsistency problem"**

**......**

**######**

**######**

**######**

**}**

If very few lines of code requires synchronisation, then its never recomeneded to delcare the entire method as synchrnoized. we have to enclose only those few lines of the code as synchrnoized with synchronized block.

The main advantage of synchronized block over synchronized method is it reduces the waiting time of the threads and it improves the performance of the system.

**Scenarios**

**=========**

**1.**

To get a lock of current object, we can declare synchronized block as follows

If a thread gets a lock of current object only then it is allowed to executed this block.

synchronized(this){

}

**2.**

To get a lock of particular object B , we can declare synchronized block as follows

If a thread gets a lock of object b only then it is allowed to executed this block.

synchronized(b){

}

**3.**

To get class lock we have declare synchrnoized block as follows

If a thread gets a lock of Display only then it is allowed to execute this block.

synchronized(Display.class){

}

**4.**

Lock concept is applicable only at object/class level it is not applicable at primtive level

int x=10;

synchronized(x){//CE: requried: referece found: primitve

}

**eg#1.**

**class Display{**

**public void wish(String name){**

**;;;;;;;;;;;;;;;;;;;;;; //1lakh line code**

**synchronized(this){**

**for (int i=1;i<=10 ;i++ ){**

**System.out.print("Good morning :");**

**try{**

**Thread.sleep(2000); //2sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println(name);**

**}**

**}**

**;;;;;;;;;;;;;;;;;;;;;; //1lakh line code**

**}**

**}**

**class MyThread extends Thread{**

**Display d;**

**String name;**

**MyThread(Display d,String name){**

**this.d=d;**

**this.name=name;**

**}**

**@Override**

**public void run(){**

**d.wish(name);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Display d= new Display();**

**MyThread t1= new MyThread(d,"dhoni");**

**MyThread t2= new MyThread(d,"yuvi");**

**t1.start();**

**t2.start();**

**}**

**}**

Output:: regular

3 Thread

a. main

b. Thread-0

execute 1 lakh lines of code,get the lock and print good moring:dhoni

(10times)

c. Thread-1

execute 1 lakh lines of code, wait till the other thread releases the lock,once

gets the lock print good morning:yuvi(10times)

**eg#2.**

**class Display{**

**public void wish(String name){**

**;;;;;;;;;;;;;;;;;;;;;; //1lakh line code**

**synchronized(this){**

**for (int i=1;i<=10 ;i++ ){**

**System.out.print("Good morning :");**

**try{**

**Thread.sleep(2000); //2sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println(name);**

**}**

**}**

**;;;;;;;;;;;;;;;;;;;;;; //1lakh line code**

**}**

**}**

**class MyThread extends Thread{**

**Display d;**

**String name;**

**MyThread(Display d,String name){**

**this.d=d;**

**this.name=name;**

**}**

**@Override**

**public void run(){**

**d.wish(name);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Display d1= new Display();**

**Display d2= new Display();**

**MyThread t1= new MyThread(d1,"dhoni");**

**MyThread t2= new MyThread(d2,"yuvi");**

**t1.start();**

**t2.start();**

**}**

**}**

Output:: irrregular

3 Thread

a. main

b. Thread-0

execute 1 lakh lines of code,get the lock of d1 and print good moring:dhoni

(10times)

c. Thread-1

execute 1 lakh lines of code, gets the lock of d2 and print good

morning:yuvi(10times)

**eg#3**

**class Display{**

**public void wish(String name){**

**;;;;;;;;;;;;;;;;;;;;;; //1lakh line code**

**synchronized(Display.class){**

**for (int i=1;i<=10 ;i++ ){**

**System.out.print("Good morning :");**

**try{**

**Thread.sleep(2000); //2sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println(name);**

**}**

**}**

**;;;;;;;;;;;;;;;;;;;;;; //1lakh line code**

**}**

**}**

**class MyThread extends Thread{**

**Display d;**

**String name;**

**MyThread(Display d,String name){**

**this.d=d;**

**this.name=name;**

**}**

**@Override**

**public void run(){**

**d.wish(name);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Display d1= new Display();**

**Display d2= new Display();**

**MyThread t1= new MyThread(d1,"dhoni");**

**MyThread t2= new MyThread(d2,"yuvi");**

**t1.start();**

**t2.start();**

**}**

**}**

Output:: regular

3 Thread

a. main

b. Thread-0

execute 1 lakh lines of code,get the classlevel lock display and print

good moring:dhoni (10times)

c. Thread-1

execute 1 lakh lines of code, wait for the class leve lock after getting class

level that particular thread will apply class level lock and starts printing

good morning:yuvi (10times)

**eg#4.**

**int x=10;**

**synchronized(x){//CE: required:reference found:int**

**for (int i=1;i<=10 ;i++ ){**

**System.out.print("Good morning :");**

**try{**

**Thread.sleep(2000); //2sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println(name);**

**}**

**}**

**Interview Questions**

**===================**

1. Explain about synchronized keyword, where we can apply? and its advantages and disadvantages?

2. What is object lock and when a Thread required?

3. What is class level lock and when a Thread required?

4. What is the difference between object lock and class level lock?

5. While a Thread executing a synchronized method on the given object is the

remaining Threads are allowed to execute other synchronized methods

simultaneously on the same object?

6. What is synchronized block and explain its declaration?

7. What is the advantage of synchronized block over synchronized method?

8. What is racecondition?

Ans: If the two or more threads tries to access the same object simultaneously

then it would result in "data inconsistency problem",This problem is

only known as "RaceCondition". To avoid this we use "synchronized"

keyword.

9. Is a Thread can hold more than one lock at a time?

Ans: yes,but lock will be of different object.

10.What is synchronized statement?

Ans::statements which are part of synchronized block and synchronized

methods are called as "synchronized statement".

eg:: sychronized(this){.....//stmt-1}

public sychronized void m1(){....//stmt-1}

Date: 06/01/2022

Topic: MultiThreading-11

========================

**InterthreadCommunication(wait,notify,notifyAll)**

1. Two threads can communicate with each other using wait(),notify(),notifyAll()

2. The thread which needs updation will call wait() on the required object, so that

a particular thread will enter into waiting state.

3.The thread which performs updation on object,it is responsible to call notify(),

notifyAll() after the notification is sent a paritcular thread will get the updation

so from the waiting state he can wake up and use the updated result.

**Note::**

wait(),notify(),notifyAll() are a part of Object class not a part of Thread class becoz notify() notifyAll(),wait() are called on objects, they are not specific to a thread.

Since these methods should be called on "Object" it should be available to all the objects in java, since it should be available it is a part of Object class not a part of Thread class.

**Keypoints about wait(),notify(),notifyAll()**

1. To call wait(),notify(),notifyAll() compulsorily the thread should be the owner of the object A thread which calls wait(),notify(),notifyAll() should get the lock of the current object.

To get the lock of a thread, that particular thread should be in synchronized

area,otherwise it would result in runtime exception called

"IllegalMonitorStateException".

1. Once a thread calls wait() method on the given object,immediately it will

release the lock and it enteres into waiting state.

(he will be waiting till he gets the notification)

1. Thread which call notify(),notifyAll() may or may not release the lock

immediately after the updation.

**d.** The only methods which are capable of releasing the lock on the thread is

wait(),notify() and notifyAll() apart from these methods lock can't be released

by any other methods.

Date: 07/01/2022

Topic: MultiThreading-12

========================

**InterthreadCommunication(wait,notify,notifyAll)**

**Method prototype of wait,notify,notifyAll**

**===================================**

public final void wait() throws InterruptedException;

public final native void wait(long ms) throws InterruptedException;

public final void wait(long ms, int ns) throws InterruptedException;

public final native void notify();

public final native void notifyAll();

**Question based on lock**

**======================**

1. If a thread calls wait() immediately it will enter into waiting state without

releasing any lock. **(invalid)**

1. If a thread calls wait() it releases the lock of that object but may not

Immediately. **(invalid)**

1. If a thread calls wait() on any object,it releases all locks acquired by that thread and enters into waiting state.**(invalid)**
2. If a thread calls wait() on any object,it immediately releases the lock of that

particular object and entered into waiting state. **(valid)**

1. If a thread calls notify() on any object,it immediately releases the lock of that particular object. **(invalid)**
2. If a thread calls notify() on any object,it releases the lock of that object but may not immediately. **(invalid)**

**eg#1.**

**class ThreadB extends Thread{**

**int total=0;**

**@Override**

**public void run(){**

**for (int i=1;i<=100;i++ ){**

**total+=i;**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args)throws InterruptedException{**

**ThreadB b=new ThreadB();**

**b.start();**

**stmt-1;**

**System.out.println(b.total);**

**}**

**}**

**If stmt-1 is replaced with Thread.sleep(10000)**

=> If main thread is made to sleep for 10sec then there is not Datainconsitency

problem,but within 1ns only the total variable value is updated by Thread-0,

but still main thread should wait for 10sec,which decreases the performance

so not a good approach to used sleep().

**eg#2.**

**class ThreadB extends Thread{**

**int total=0;**

**@Override**

**public void run(){**

**for (int i=1;i<=100;i++ ){**

**total+=i;**

**}**

**}**

**;;;;;;;;;;;;;;;;; //1cr lines of code**

**}**

**public class TestApp {**

**public static void main(String[] args)throws InterruptedException{**

**ThreadB b=new ThreadB();**

**b.start();**

**stmt-1;**

**System.out.println(b.total);**

**}**

**}**

**=> if stmt-1 is replaced with b.join()**

Till the execution of all the statement present in run() the main thread should

wait,but main thread is expecting only the updated value from the object, that

updation will happen with in 1ns,but main thread should wait till the other

thread completes the execution.

This would decrease the peformance so "join()" is not suited.

**eg#3.**

**class ThreadB extends Thread{**

**int total=0;**

**@Override**

**public void run(){**

**synchronized(this){**

**System.out.println("Child thread is updating the variable");**

**for (int i=1;i<=100;i++ ){**

**total+=i;**

**}**

**System.out.println("Child thread is notifying");**

**this.notify();**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args)throws InterruptedException{**

**ThreadB b=new ThreadB();**

**b.start();**

**synchronized(b){**

**System.out.println("Main thread waiting for updation");**

**b.wait();**

**System.out.println("Main thread got notification");**

**System.out.println(b.total);**

**}**

**}**

**}**

Output

Main thread waiting for updation

Child thread is updating the variable

Child thread is notifying

Main thread got notification

5050

Date: 07/01/2022

Topic: MultiThreading-12

========================

**InterthreadCommunication(wait,notify,notifyAll)**

**eg#1.**

**class ThreadB extends Thread{**

**int total=0;**

**@Override**

**public void run(){**

**synchronized(this){**

**System.out.println("Child thread is updating the variable");**

**for (int i=1;i<=100;i++ ){**

**total+=i;**

**}**

**System.out.println("Child thread is notifying");**

**this.notify();**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args)throws InterruptedException{**

**ThreadB b=new ThreadB();**

**b.start();**

**Thread.sleep(10000);//10sec**

**synchronized(b){**

**System.out.println("Main thread waiting for updation");**

**b.wait();**

**System.out.println("Main thread got notification");**

**System.out.println(b.total);**

**}**

**}**

**}**

Output::

2 Threads

a. main

1. main thread enters into sleeping state,so TS assigns Thread-0

2. After sleeping for 10sec immediately woke up to get the notification

3. It enters into waiting state and remains in waiting state for ever

b. Thread-0

run the loop and produced the result as 5050,notified the waiting thread.

Thread-0 is finished with the execution so it enters into dead state

Child thread is updating the variable

Child thread is notifying

Main thread waiting for updation

//waiting for ever to get the notification

**eg#2.**

**class ThreadB extends Thread{**

**int total=0;**

**@Override**

**public void run(){**

**synchronized(this){**

**System.out.println("Child thread is updating the variable");**

**for (int i=1;i<=100;i++ ){**

**total+=i;**

**}**

**System.out.println("Child thread is notifying");**

**this.notify();**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args)throws InterruptedException{**

**ThreadB b=new ThreadB();**

**b.start();**

**Thread.sleep(10000);//10sec**

**synchronized(b){**

**System.out.println("Main thread waiting for updation");**

**b.wait(10000);//10sec**

**System.out.println("Main thread got notification");**

**System.out.println(b.total);**

**}**

**}**

**}**

Output

2 Threads

a. main

1. main thread enters into sleeping state,so TS assigns Thread-0

2. After sleeping for 10sec immediately woke up to get the notification

3. It enters into waiting state and waits only for 10sec to get notification

4. with in 10sec time no notification means it will continue its execution

b. Thread-0

run the loop and produced the result as 5050,notified the waiting thread.

Thread-0 is finished with the execution so it enters into dead state

Child thread is updating the variable

Child thread is notifying

Main thread waiting for updation

Main thread got notification

5050

**Difference b/w notify and notifyAll()**

notify() => It is used to give notification only for one thread

notifyAll() => It is used to give notification for many thread

notify() => When there are many threads which are waiting the notify() will notify

all the threads on that particular object, who are waiting so all will be

notified.Which thread will get a chance can't be decided it is upto the

ThreadScheduler.

notifyAll()=> We can use notifyAll() method to give the notification for all

waiting Threads of particular object.

All waiting Threads will be notified and will be executed one by

one, because they required lock.

**Note::**

On which object we are calling wait(), notify() and notifyAll() methods that

corresponding object lock we have to get but not other object locks.

Date: 11/01/2022

Topic: MultiThreading-13

==========================

**Deadlock**

**=======**

* If two threads are waiting for each other forever, that scenario we call it as

"DeadLock".

* There are no resolution techniques available for deadlock, we have only

prevention technique.

* synchronized keyword is creating a problem of deadlock,so we need to be careful while using synchronized keyword.

**eg#1.**

**class A {**

**public void d1(B b){**

**System.out.println("Thread-1 started executing d1()");**

**try{**

**Thread.sleep(5000);//5sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println("Thread-1 trying to call b.last()");**

**b.last();**

**}**

**public void last(){**

**System.out.println("A class last() method");**

**}**

**}**

**class B {**

**public void d2(A a){**

**System.out.println("Thread-2 started executing d2()");**

**try{**

**Thread.sleep(5000);//5sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println("Thread-2 trying to call a.last()");**

**a.last();**

**}**

**public void last(){**

**System.out.println("B class last() method");**

**}**

**}**

**public class TestApp extends Thread{**

**A a =new A();**

**B b =new B();**

**public void m1(){**

**this.start();**

**a.d1(b);//executed by main thread(main)**

**}**

**@Override**

**public void run(){**

**b.d2(a);//executed by child thread(Thread-0)**

**}**

**public static void main(String[] args){**

**TestApp t=new TestApp();**

**t.m1();**

**}**

**}**

Output(No Sychrnoized, so no deadlock)

2 threads

a. main

b. Thread-0

Thread-1 started executing d1()

Thread-2 started executing d2()

Thread-1 trying to call b.last()

Thread-2 trying to call a.last()

B class last() method

A class last() method

**eg#2.**

**class A {**

**public synchronized void d1(B b){**

**System.out.println("Thread-1 started executing d1()");**

**try{**

**Thread.sleep(5000);//5sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println("Thread-1 trying to call b.last()");**

**b.last();**

**}**

**public void last(){**

**System.out.println("A class last() method");**

**}**

**}**

**class B {**

**public synchronized void d2(A a){**

**System.out.println("Thread-2 started executing d2()");**

**try{**

**Thread.sleep(5000);//5sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println("Thread-2 trying to call a.last()");**

**a.last();**

**}**

**public void last(){**

**System.out.println("B class last() method");**

**}**

**}**

**public class TestApp extends Thread{**

**A a =new A();**

**B b =new B();**

**public void m1(){**

**this.start();**

**a.d1(b);//executed by main thread(main)**

**}**

**@Override**

**public void run(){**

**b.d2(a);//executed by child thread(Thread-0)**

**}**

**public static void main(String[] args){**

**TestApp t=new TestApp();**

**t.m1();**

**}**

**}**

Output(Sychrnoized, still no deadlock because it is used properly)

2 threads

a. main

b. Thread-0

Thread-1 started executing d1()

Thread-2 started executing d2()

Thread-2 trying to call a.last()

A class last() method

Thread-1 trying to call b.last()

B class last() method

**eg#3.**

**class A {**

**public synchronized void d1(B b){**

**System.out.println("Thread-1 started executing d1()");**

**try{**

**Thread.sleep(5000);//5sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println("Thread-1 trying to call b.last()");**

**b.last();**

**}**

**public synchronized void last(){**

**System.out.println("A class last() method");**

**}**

**}**

**class B {**

**public synchronized void d2(A a){**

**System.out.println("Thread-2 started executing d2()");**

**try{**

**Thread.sleep(5000);//5sec**

**}**

**catch (InterruptedException ie){**

**}**

**System.out.println("Thread-2 trying to call a.last()");**

**a.last();**

**}**

**public synchronized void last(){**

**System.out.println("B class last() method");**

**}**

**}**

**public class TestApp extends Thread{**

**A a =new A();**

**B b =new B();**

**public void m1(){**

**this.start();**

**a.d1(b);//executed by main thread(main)**

**}**

**@Override**

**public void run(){**

**b.d2(a);//executed by child thread(Thread-0)**

**}**

**public static void main(String[] args){**

**TestApp t=new TestApp();**

**t.m1();**

**}**

**}**

Output(Sychrnoized, deadlock because it is not used properly)

2 threads

a. main

b. Thread-0

Thread-1 started executing d1()

Thread-2 started executing d2()

Thread-1 trying to call b.last()

Thread-2 trying to call a.last()

**DaemonThread**

**===========**

The threads which are executing in the background are called as "DaemonThread".

Objective of DaemonThead these Daemon thread will support for the execution

of "NonDaemonThread".

eg:: GarbageCollector thread will always run in the background to support main

thread such that memory shortage should not happen for the main thread in

execution.

**Note::**

1. Using ThreadApi,it is possible to check whether the thread is Daemon or not.

a. public boolean isDaemon()

1. For every userdefined thread,its deamon natrue will be coming from parent, if the parent thread is nonDaemon, then child thread also will be NonDaemon only by default.

But it is possible to make the a particular thread as Daemon by using a

method called setDaemon(true).

a. public void setDaemon(boolean b)

b => true it becomes Daemon.

**Note::**

Before starting the thread only if we want to change the nature of a

particular thread as Daemon, we should change, if we try to change

once the thread starts it would result in "IllegalThreadStateException".

3. Is it possible to make main thread as "Daemon thread"

Ans: No, because starting of main thread is not in the hands of the

programmer,it is in the hands of "JVM", so we cannot change the JVM

code of starting main thread so main thread can't be made as

"DaemonThread".

**eg::**

**class MyTrhead extends Thread{}**

**class TestApp{**

**public static void main(String... args){**

**System.out.println(Thread.currentThread().isDaemon());//false**

**MyThread t=new MyThread();**

**System.out.println(t.isDaemon());//false**

**t.start();**

**t.setDaemon(true);//IllegalThreadStateException**

**System.out.println(t.isDaemon());**

**}**

**}**

4. How long will DaemonThreads support the NonDaemonThreads?

Once the NonDeamon thread finishes the execution, DaemonThread life is

also gone.

**eg#1.**

**class MyThread extends Thread{**

**public void run(){**

**for (int i=1;i<=10 ;i++ ){**

**System.out.println("child thread");**

**try{**

**Thread.sleep(2000);//2sec**

**}**

**catch (InterruptedException ie){**

**}**

**}**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**MyThread t=new MyThread();**

**t.setDaemon(true);//stmt-1**

**t.start();**

**System.out.println("end of main thread");**

**}**

**}**

**if stmt-1 is commented, how many threads and how many DaemonThreads**

Ans. 2 threads both are nonDaemon

a.main

end of main thread

b.Thread-0

child thread will continue to print(10 times)

**if stmt-1 is t.setDaemon(true), how many threads and how many DaemonThreads?**

Ans. 2 threads

a.main(NonDaemon)

end of main thread(once the NonDaemon thread finishes the job,

Daemon thread life cycle also ends)

b.Thread-0(Daemon)

collection FrameWork

Date: 13/01/2022

**CollectionFramework**

**=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=\*=**

**Variable approach:**

In case of varaible approach using one varaible we can hold only one value, to

resolve the problem of one varible holding mulitple value we use the concept

of "Arrays".

**Array**

=> It refers to an indexed collection of homogenous data elements

=> Advnatage of array is single variable can hold mulitple values

DisAdvnantage

============

1.If the size of the array is fixed, then we cannot increase nor decrease the size

of an array at the runtime

2.It can store only homogenous data elements

eg:: int[] arr =new int[5];//only int values

Student[] s=new Student[5]; // only student objects

3.Array concept is not implemented using any of the standard datastructure,due

to which as a programmer to work with array we need to implement

logic,which increases the complexity of programming.

**To resolve the above mentioned problem we use "Collection".**

Advantages of Collection

========================

1. Collection are growable in nature, we can increase or decrease the size

dynamically.

1. Collection can hold both homogenous and heterogenous elements.
2. Every collection class is implemented using standard datastructure, so being a programmer to work with collection class we need not implement any logic already logic is a part of the class(API) so complexity of

programming is reduced.

Collection we choose when we want an application to use the memory efficiently

Array we choose when we want our application to be best at the performance level.

**Difference b/w Array and Collection**

**=============================**

Array => It is fixed in size

Collection => It is dynamic in nature

Array => It can hold only homogenous data elements

Collection => It can hold both homogenous and heterogenous data elements

Array => It is not implemented using any data strucutred internally

Collection => Every collection class is internally implemented using standard

datastructure

Array => Memory wise Array are not recomended

Collection => Memory wise Collection are recomended

Array => Performance wise Array is recomended

Collection => Performance wise Collection is recomended

Array => It can hold both primtive and object type

Collection => It can hold only object,but not primitive type

**Collection**

**===========**

In order to represent a group of individual objects as a single entity we use Collection

**CollectionFramework**

**===================**

To store group of individual objects as a single entity we use few inbuilt classes

those classes are a part of API, which we call it as "CollectionFramework".

**API** => It refers to Collection of .class files(interface + classes + enum)

Collection and CollectionFramework is not only a part of java language,it was there in C++ also with different names.

**Note::**

**=====**

JAVA C++

Collection container

CollectinFramework STL(standard template library)

**9 Interfaces are a part of CollectionFramework**

**==================================**

1. Collection(I)

2. List(I)

3. Set(I)

4. SortedSet(I)

5. NavigableSet(I)

6. Queue(I)

7. Map(I)

8. SortedMap(I)

9. NavigableMap(I)

**Collection(I):**

* In order to represent group of individual objects as a single entity we go for Collection(I).
* It is a root interface for all Collection framework.
* It defines the common methods which are required for all Collection Object.

**Note::** There is not direct implementation class for Collection(I)

**Interview Question**

**==================**

1. Difference b/w Collection and Collections?

Collection => It is an interface which is used to hold mulitple objects as a

single entity

Collections => It is an utility class which is a part of java.util package, through

Which it supports the user to write less code and do more work.

eg:: Collections.sort(al);//it sorts the data in ascending order.

Date: 17/01/2022

Topic: CollectionFramework

==========================

**9 Interfaces are a part of CollectionFramework**

**==================================**

1. Collection(I)

2. List(I)

3. Set(I)

4. SortedSet(I)

5. NavigableSet(I)

6. Queue(I)

7. Map(I)

8. SortedMap(I)

9. NavigableMap(I)

**Collection(I)**

* It is a root interface for all the other Collection Classes/interfaces
* It defines all the common methods which is required for all the other Collection/interfaces.
* It is prefered when we want to represent a group of individual objects as a single entity.

**Note::** There is no implementation class directly available for Collection(I).

**List(I)**

1. It is a child interface of Collection

2. To represent a group of individual object as a single entity where

a. duplicates are permitted

b. insertion order to be preserved we opt for List(I)

3. Its implementation classes are ArrayList,LinkedList.

4. Legacy classes implementation of List are Vector,Stack.

**Set(I)**

1. It is a child interface of Collection

2. To represent a group of individual object as a single entity where

a.duplicates are not allowed

b.insertion order is not preserved

3. Its implementation classes are HashSet,LinkedHashSet

**SortedSet(I)**

1. It is a child interface of Set

2. To represent a group of individual object as a single entity where

a. elemenents should be stored in some sorted sorder we prefere using

sortedSet

**NavigableSet(I)**

1. It is the child interface of SortedSet

2. This interface define few specific method which are related for navigation

Purpose.

3. Tree set is the implementation class for Navigable Set.

**Queue(I)**

1. It is the child interface of Collection

2. To represent group of individual object prior to processing we need to use

Queue

**eg::** Before sending a mail to our client, we need to store all the client details in

any one of our datastructure, As how the client details are added same way

the mail should be triggered,for this requriement best suited is Queue(FIFO).

All the above mentioned interfaces and classes are meant for holding the objects as a singl entity,but if we want to store the object details as key-value pair then we need to opt for "Map".

**Map(I)**

1. It is not a child interface of Collection

2. If we want to represent a group of individual object as a key value pair then

we need to opt for Map(I).

3. Duplicate keys are not permitted whereas values are permitted.

* Map(I) => Store the data as key value pair then choose Map(I).
* SortedMap(I) => To store the data in the key value pair, where sorting

should happen based on "key".

* NavigableMap(I) => On the Map object, if we want some navigable

options then we opt for "NavigableMap".

**Note::** refer class notes for hierarcy chart.

Date: 18/01/2022

Topic: CollectionFramework

==========================

**9 Interfaces are a part of CollectionFramework**

**===================================**

1. Collection(I)

2. List(I)

3. Set(I)

4. SortedSet(I)

5. NavigableSet(I)

6. Queue(I)

7. Map(I)

8. SortedMap(I)

9. NavigableMap(I)

**Collection(I)**

* It is a root interface for all the other Collection Classes/interfaces
* It defines all the common methods which is required for all the other Collection/interfaces.
* It is prefered when we want to represent a group of individual objects as a single entity.

Inside this interface, the commonly used method required for all the collection

classes is present.

**a.** **boolean add(object o)** => Only one object

**b.** **boolean addAll(Collection c)** =>To add group of Object

**c.** **boolean remove(Object o)** => to remove particular object

**d.** **boolean removeAll(Collection c)**=> to remove particular group of collection

**e.** **void clear()** => to remove all the object

**f.** **int size()** => to check the size of the Collection

**g. boolean retainAll(Collection c)** => except this group of objects remaining all

objects should be removed.

**h. boolean contains(Object o)** => to check whether a particular object exists or

not.

**i. boolean containsAll(Collection c)** => To check whether a particular

collection exists or not

**j.** **boolean isEmpty()** => To check whether the Collection is empty or not

**k.** **Object[] toArray()**=> Convert the object into Array.

**l.** **Iterator iterator()** => cursor need to iterate the collection object

**To iterate the elements from the Collection we have 3 cursors**

a. Enumeration

b. Iterator

c. ListIterator

**Note::** For Collection class, there is no direct concrete class which gives the

implementation.

**List::**

**=\*=\*=**

1.In order to store group of individual object as a single entity

a. duplicates are allowed

b. insertion order is preserved

then we opt for List

**Commond methods**

**================**

a. **void add(int index,Object obj)**

b. **void addAll(int index,Collection c)**

c. **Object remove(int index)**

d. **Object get(int index)**

This method is used to get the elements from the respective index

e. **Object set(int index,Object o)**

This method is used to replace a particular object at the respective index.

if the element already existed in that index,it will be pushed to the right.

f. **int indexOf(Object obj)**

g. **int lastIndexOf(Object obj)**

i. **ListIterator listIterator()**

**ArrayList**

**=========**

Datastructure :Resizable Array/Growable Array

Heterogeneous :Yes it allows

Duplicates :yes it allows

insertion order :yes it preserves

null insertion :yes it allows

**Different ways of Creating ArrayList**

**============================**

a. **ArrayList al =new Arraylist();**

//Default capacity is 10

//Next capacity will be calculated based on :: (currentcapcity \* 3/2) + 1

b. **ArrayList al =new ArrayList(int initalCapacity);**

c. **ArrayList al =new ArrayList(Collection c);**

**ArrayList:**

**Datastructure::**  Growable Array/Resizable Array

**Duplicates ::** yes allowed(through index)

**insertion order preserved::** yes(through index)

**null insertion allowed ::** yes

**eg#1.**

**import java.util.ArrayList;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al = new ArrayList();**

**al.add("A");**

**al.add(10);**

**al.add("A");**

**al.add(null);**

**System.out.println(al);**//[A,10,A,null]

**al.remove(2);**

**System.out.println(al);** //[A,10,null]

**al.add(2,'M');**

**System.out.println(al);** //[A,10,M,null]

**al.add('M');**

**System.out.println(al);** //[A,10,M,null,M]

**}**

**}**

Whenever we print reference,internally jvm will call toString() on that reference.

In all Collection implementation classes, the code of toString() is overriden to print the objects in the following format .

**[obj1,obj2,obj3,.....]**

**Note::** Usally we use collection to hold the object and those objects we need to

transmit over the network, if a particular object has to be transported over

the n/w then those object should get the facility of "Serializalbe"

=> By default ArrayList class implements Serializable and Cloneable Interface.

=> To support RandomAccess of elements it also implements an interface called

"RandomAccess".

=> ArrayList and Vector are the only implementation classes of

"RandomAccess".

**eg#1.**

**import java.util.ArrayList;**

**import java.io.Serializable;**

**import java.util.RandomAccess;**

**import java.util.LinkedList;**

**import java.util.Vector;**

**import java.lang.String;**

**import java.lang.System;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al = new ArrayList();**

**LinkedList ll=new LinkedList();**

**Vector v=new Vector();**

**System.out.println(al instanceof Serializable);//true**

**System.out.println(al instanceof Cloneable);//true**

**System.out.println(al instanceof RandomAccess);//true**

**System.out.println();**

**System.out.println(ll instanceof Serializable);//true**

**System.out.println(ll instanceof Cloneable);//true**

**System.out.println(ll instanceof RandomAccess);//false**

**System.out.println();**

**System.out.println(v instanceof Serializable);//true**

**System.out.println(v instanceof Cloneable);//true**

**System.out.println(v instanceof RandomAccess);//true**

**}**

**}**

=> ArrayList is best suited if the frequent operation is retrival

=> ArrayList is not good if the frequent operation is

a. inserting the element in b/w

b. removing the element based on the index

because it needs so many shifts which reduces the performances

=> To resolve this problem we need to use "LinkedList".

Difference b/w Collection vs Collections?

Difference b/w List vs Set?

Difference b/w ArrayList vs Vector?

ArrayList =>Methods are not synchronized

Vector =>Methods are synchronized

ArrayList => since it is not synchronized, the object is not thread safe

Vector => since it is synchronized, the object is thread safe

ArrayList => Since it is not synchronized, performance is high(no waiting

threads)

Vector => Since it is synchronized, perforamnce is low(threads will be in

waiting state)

ArrayList => realeased in 1.2v

Vector => legacy class(1.0v)

ArrayList methods are non-synchronized,How to get synchronized version of ArrayList?

=> To get synchronized version of ArrayList, we take the help of Utility class

called "Collections"

**eg#1.**

**import java.util.List;**

**import java.util.ArrayList;**

**import java.util.Collections;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al = new ArrayList();//non-synchronized version**

**List l = Collections.synchronizedList(al);**

**}**

**}**

al => nonsychrnoized version

l => synchronized version

**Note::**

Method prototype

**public static List synchronizedList(List l)**

similary we also have it to for set and map as shown below

**public static Set synchronizedList(Set s)**

**public static Map synchronizedList(Map m)**

* ArrayList is best suited only when our frequent operation is retrieval
* ArrayList is not suited when we want to insert the element in middle/delete the element in middle.

to resolve this problem we need to use **"LinkedList".**

**LinkedList**

**==========**

**DataStructure:** doublylinkedlist

**heterogenous :** yes allowed

**null :** yes allowed

**duplicates :** yes allowed

**insertionorder :** preserved

* It implements Serializable,Cloneable interface but not RandomAccess.
* It is best suited when our frequent operation is "insertion/deletion in middle".
* It is not suited when our frequent operation is "retrieval".

**Constructors of LinkedList**

**======================**

**LinkedList l=new LinkedList()**

**LinkedList l=new LinkedList(Collection c)**

Using linkedList making it to work like stack and queue is a common

requirement,to support this

LinkedList class has given few methods as shown below

**1. public void addFirst(Object o);**

**2. public void addLast(Object o);**

**3. public Object getFirst();**

**4. public Object getLast();**

**5. public Object removeFirst();**

**6. public Object removeLast();**

**eg#1.**

**import java.util.LinkedList;**

**public class TestApp {**

**public static void main(String[] args){**

**LinkedList ll=new LinkedList();**

**ll.add("sachin");**

**ll.add(10);**

**ll.add(null);**

**ll.add("sachin");**

**System.out.println(ll);//**[sachin,10,null,sachin]

**ll.set(0,"kohli");**

**System.out.println(ll);//**[kohli,10,null,sachin]

**ll.removeLast();**

**System.out.println(ll);//**[kohli,10,null]

**ll.addFirst("sachin");**

**System.out.println(ll);//**[sachin,kohli,10,null]

**}**

**}**

**Vector**

**=====**

**DataStructure :** GrowableArray/Resizable Array

**heterogenous :** yes allowed

**null :** yes allowed

**duplicates :** yes allowed

**insertionorder :** preserved

* It implements Serializable,Cloneable interface and also RandomAccess.
* It is best suited when our frequent operation is "retrieval".
* It is not suited when our frequent operation is "insertion/deletion".
* Most of the methods of Vector is "synchronized" so it is "Thread safe".
* It is a legacy class(1.0V)

Since the class is a legacy class, the method names of those class is lengthy.

**Add element**

a. add(Object o) => Collection

b. add(int index,Object o) => List

c. addElement(Object o) => Vector

--------------------------------------

**Remove element**

a. remove(Object o) => Collection

b. removeElement(Object o) => Vector

----------------------------------

a. remove(int index) => List

b. removeElementAt(int index) => Vector

-----------------------------------

a. clear() => Collection

b. removeAllElements() => Vector

**Retrieve elements**

a. Object get(int index) => List

b. Object elementAt(int index) => Vector

c. Object firstElement() => Vector

d. Object lastElement() => Vector

**Constructors available in Vector**

**=========================**

1. **Vector v=new Vector();**

a. Defalut capacity is 10

b. if capacity is full, it grows in size with a capacity as (oldcapacity \*2)

2. **Vector v=new Vector(int capacity);**

3. **Vector v=new Vector(int capacity,int incrementalcapacity)**

To use the memory effectively,we take the help of incremental capacity.

a. Vector will be created with given capacity

b. if the capacity if full, then vector will not grow by double, it will grow

based on incremental capacity.

4. **Vector v=new Vector(Collection c);**

**Curosrs in Collection Framework**

**=========================**

1. Enumeration(I)

2. Iterator(I)

3. ListIterator(I)

**Utility Classes**

**=============**

1. Arrays

2. Collections

**Based on Sorting to happen**

**======================**

1. Comparable(I)=> default natural sorting order

2. Comparator(I)=> customized sorting order

**Vector example**

**==============**

**import java.util.Vector;**

**public class TestApp {**

**public static void main(String[] args){**

**Vector v= new Vector();**

**System.out.println("Default capacity is :: " +v.capacity());//10**

**for (int i=0;i<=9; i++){**

**v.addElement(i);**

**}**

**System.out.println("Capacity after adding elements is :: " +v.capacity());//10**

**v.addElement("sachin");**

**System.out.println("Increased capacity is :: " +v.capacity());//20**

**System.out.println(v);//[0,1,2,3,4,5,6,7,8,9,sachin]**

**}**

**}**

**Stack**

**=====**

1. It is a child class of Vector

2. This class is specially designed to follow LIFO/FILO order

**Constructor ::** Stack s=new Stack()

Common operation associated with stack are push,pop,peek,empty,search

1**. Object push(Object o)** => Push the element into stack

2**. Object pop()** => remove the last element from the stack

3**. Object peek()** => returns the top element of the stack,without removal

4**. boolean empty()** => checks whether the stack is empty or not

5**. int search(Object o)** => It checks for the specified object and it returns the

offset if found

otherwise it returns -1.

**eg#1.**

**import java.util.Stack;**

**public class TestApp {**

**public static void main(String[] args){**

**Stack s=new Stack();//Default Capacity is :10**

**s.push("A");**

**s.push("B");**

**s.push("C");**

**System.out.println(s);//[A,B,C]**

**s.pop();**

**System.out.println(s);//[A,B]**

**s.push("C");**

**System.out.println(s);//[A,B,C]**

**System.out.println(s.search("C"));//1(offset it returns)**

**System.out.println(s.search("sachin"));//-1**

**}**

**}**

**Cursor in CollectionFramework**

**========================**

In order to retrieve the object from the collection one by one we need to use Cursors

There are 3 cursors in Collection Framework

a. Enumeration(legacy cursor)

b. Iterator(universal cursor)

c. ListIterator

**Enumeration**

**===========**

1. It is a cursor which can be used on legacy classes

2. It can be used to get the collection object one by one.

3. It has 2 methods

a. hasMoreElements=> checks whether still object is there are not if found

returns true otherwise false.

b. nextElement => returns the Object at the current position and change the

cursor to the next object.

**eg#1.**

**import java.util.Vector;**

**import java.util.Enumeration;**

**public class TestApp {**

**public static void main(String[] args){**

**Vector v= new Vector();**

**for (int i=1;i<=10 ;i++ ){**

**v.addElement(i);**

**}**

**System.out.println(v);//**[1,2,3,4,5,6,7,8,9,10]

**System.out.println("Accessing through enumeration");**

**Enumeration e= v.elements();**

**while(e.hasMoreElements()){**

**Integer i =(Integer)e.nextElement();**

**if (i%2==0)**

**System.out.println(i);**

**}**

**}**

**}**

**Limitations**

1. It is applicable only for Legacy classes.

2. Using enumeration we can perform only read operation, we can't perform

remove operation.

To overcome this limitation we need to use a cursor called **"Iterator".**

**Iterator**

**=======**

* It is cursor which can be used to perform both read and remove operation.
* It is called as universal cursor becoz it can be used on any Collection object to retrieve the Object one by one.
* This iterator object can be obtained by using iterator() of Collection(I).

methods are

1. **hasNext()**=> checks whether still object is there are not if found

returns true otherwise false.

2. **next()** => returns the Object at the current position and change the cursor

to the next object.

3. **remove()** => removes the particular object where the cursor is pointing at.

**eg#1.**

**import java.util.ArrayList;**

**import java.util.Iterator;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList l=new ArrayList();**

**for (int i=1;i<=10 ;i++ ){**

**l.add(i);**

**}**

**System.out.println(l);//[1,2,3,4,5,6,7,8,9,10]**

**Iterator itr= l.iterator();**

**while (itr.hasNext()){**

**Integer i =(Integer) itr.next();**

**if (i%2==0)**

**System.out.println(i);//2 4 6 8 10**

**else**

**itr.remove();**

**}**

**System.out.println(l);//[2,4,6,8,10]**

**}**

**}**

Date: 21/01/2022

Topic: CollectionFramework

==========================

**9 Interfaces are a part of CollectionFramework**

**====================================**

1. Collection(I)

2. List(I)

3. Set(I)

4. SortedSet(I)

5. NavigableSet(I)

6. Queue(I)

7. Map(I)

8. SortedMap(I)

9. NavigableMap(I)

**Curosrs in Collection Framework**

**===========================**

1. Enumeration(I)

2. Iterator(I)

3. ListIterator(I)

**Utility Classes**

**===============**

1. Arrays

2. Collections

**Based on Sorting to happen**

**=====================**

1. Comparable(I)

2. Comparator(I)

**Drawbacks w.r.t Iterator**

**===================**

1. By using Enumeration and Iteration, we can iterate the elements only in

forward direction we can't move in backward direction so we say

Enumeration and Iterator has single direction cursor

1. using Iterator we can just perform read and remove operation oly, we can't

perfom replace and adding of new object.

To resolve this problem we need to use **ListIterator.**

**ListIterator**

**============**

* It is the child interface of Iterator
* By using ListIterator we can move in both the direction, forward and backward.
* Addition of read and remove operation we can also perform update and addition of new object to existing Collection.

How to get ListIterator Object?

**public ListIterator listIterator()**

**Methods associated with ListIterator**

**ForwardDirection**

boolean hasNext();

Object next();

int nextIndex();

===========================

**BackwardDirection**

boolean hasPrevious();

Object previous();

int previousIndex();

===========================

void remove();

void set(Object o);

void add(Object o);

**eg#1.**

**import java.util.LinkedList;**

**import java.util.ListIterator;**

**public class TestApp {**

**public static void main(String[] args){**

**LinkedList l=new LinkedList();**

**l.add("sachin");**

**l.add("kohli");**

**l.add("rahul");**

**l.add("dhoni");**

**l.add("saurav");**

**System.out.println(l);** //[sachin,kohli,rahul,dhoni,saurav]

**ListIterator litr= l.listIterator();**

**while (litr.hasNext()){**

**String data= (String)litr.next();**

**if (data.equals("dhoni"))**

**litr.remove();**

**if (data.equals("sachin"))**

**litr.add("tendulkar");**

**if (data.equals("rahul")){**

**litr.set("dravid");**

**}**

**}**

**System.out.println(l);** //[sachin,tendulkar,kohli,dravid,saurav]

**}**

**}**

**eg#2.**

**import java.util.LinkedList;**

**import java.util.ListIterator;**

**public class TestApp {**

**public static void main(String[] args){**

**LinkedList l=new LinkedList();**

**l.add("sachin");**

**l.add("kohli");**

**l.add("rahul");**

**l.add("dhoni");**

**l.add("saurav");**

**System.out.println(l);** //[sachin,kohli,rahul,dhoni,saurav]

**ListIterator litr= l.listIterator();**

**while (litr.hasNext()){**

**String data= (String)litr.next();**

**System.out.print(data+"\t");**//sachin kohli rahul dhoni

saurav

**}**

**System.out.println();**

**while (litr.hasPrevious()){**

**String data= (String)litr.previous();**

**System.out.print(data+"\t");**//saurav dhoni rahul kohli

sachin

**}**

**System.out.println();**

**}**

**}**

**Note::**

ListIterator is the most powerful cursor, but it can be applied only on List object but not on other Collection objects.

**Comparison b/w cursors**

**======================**

**Enumeration** => can be applied only on legacy classes

Movement is single direction

To get Object we use elements()

methods are 2(hasMoreElements(),nextElement())

operation allowed is only read.

Version : 1.0

**Iterator** => can be applied on all Collection objects

Movement is single direction

To get Object we use iterator()

methods are 3(hasNext(),next(),remove)

operation allowed is read and remove.

Version : 1.2

**ListIterator** => can be applied only on List Object.

Movement is Bi-direction

To get Object we use listiterator()

methods are 9

(hasNext(),next(),nextIndex(),remove,hasPrevious(),

previous(),previousIndex(),set(Object),add(Object))

operation allowed is read ,remove,add,replace.

Version : 1.2

**Internal implementaion of cursor**

**==========================**

**import java.util.Vector;**

**import java.util.Enumeration;**

**import java.util.Iterator;**

**import java.util.ListIterator;**

**public class TestApp {**

**public static void main(String[] args){**

**Vector v=new Vector();**

**Enumeration e= v.elements();**

**Iterator itr= v.iterator();**

**ListIterator litr= v.listIterator();**

**System.out.println("Enumeration class name is ::"+e.getClass().getName());**

**System.out.println("Iterator class name is ::"+itr.getClass().getName());**

**System.out.println("ListIterator class name is ::"+litr.getClass().getName());**

**}**

**}**

**Set(I)**

**======**

1. It is a child interface of Collection

2. If we want to represent a group of individual object as a single entity where

a. duplicates are not allowed

b. insertion order is not preserved then we need to go for Set

3. Set interface does not contain any new method rather it takes the help of

Collection interface

methods only(12 Methods).

**HashSet**

**=======**

* **Underlying DataStructure** **:** HashTable(internally uses hashing

technique(hashcode) to keep the data)

* **heterogeneous elements**  **:** Yes
* **insertion order preserved :** No
* **null insertion allowed** **:**  Yes
* **duplicates allowed** **:** No
* **interface implementing**  **:** Serializable,Cloneable
* **Best suited** **:** since the data is stored based on hashing the

searching operation is very fast.

**Constructor associated with HashSet**

**==============================**

1**.** **HashSet s=new HashSet();** /**/**Default capacity => 16

Default LoadFactor => 0.75

2**. HashSet s=new HashSet(int initialCapacity);**

|=> user specified capacity

3**. HashSet s=new HashSet(int initialCapacity,int loadFactor);**

|=> user specified capacity

|=> user specified loadFactor

4**. HashSet s=new HashSet(Collection c);**

**eg#1.**

**import java.util.HashSet;**

**public class TestApp {**

**public static void main(String[] args){**

**HashSet hs=new HashSet();**

**hs.add("B");**

**hs.add("C");**

**hs.add("D");**

**hs.add("Z");**

**hs.add(10);**

**hs.add(null);**

**System.out.println(hs.add("Z"));** //false

**System.out.println(hs);** //[null,B,C,D,Z,10] random order

can't predict

**}**

**}**

**public boolean add(Object o)**

|=> returns true, if object is added to HashTable internally otherwise it **returns**

false through which duplicates are avoided.

**LinkedHashSet**

**============**

* It is the child class of HashSet
* It exhibits almost the same behaviour of HashSet.

**Difference b/w HashSet and LinkedHashSet**

**==================================**

HashSet => Underlying datastructure is "HashTable".

LinkedHashSet => Underlying datastructure is "LinkedList + HashTable".

HashSet => Insertion order is not preserved and duplicates are not allowed

LinkedHashSet => Insertion order is preserved,duplicates are not allowed

HashSet => Introduced in 1.2V

LinkedHashSet => Introduced in 1.4V

**eg#1.**

**import java.util.LinkedHashSet;**

**public class TestApp {**

**public static void main(String[] args){**

**LinkedHashSet hs=new LinkedHashSet();**

**hs.add("B");**

**hs.add("C");**

**hs.add("D");**

**hs.add("Z");**

**hs.add(10);**

**hs.add(null);**

**System.out.println(hs.add(10));//false**

**System.out.println(hs);//[B,C,D,Z,10,null]**

**}**

**}**

**Note::**

To build cacheBased application where duplicates are not allowed and insertion

order is not important then we go for HashSet or LinkedHashSet.

**SortedSet(I)**

**==========**

1. It is the child interface of Set

2. If we want to represent a group of individual object as a single entity, where

a.duplicates are not allowed

b.insertion order is preserved based on some sorting order

then we opt for SortedSet.

3. The default natural sorting order of Numbers is "Ascending order".

The default natrual sorting order of String is "Alphabetical order".

4. The sorting can be made as Default natural sorting order or we can use

"Customized Sorting order".

**Methods associated with SortedSet**

**===============================**

1. **Object first()** => it returns the first element in the set

2. **Object last()** => it retursn the last element in the set

3. **SortedSet headSet(Object o)** => it returns the element < obj

4. **SortedSet tailSet(Object o)** => it returns the element > = obj

5. **SortedSet subset(Object o1,Object o2)** => it returns the

element **>=**obj1 but less than obj2

7. **Compartor comparator** => If the elements are arranged in some default

natural sorting order then it returns null,

otherwise it returns the underlying sorted

order mechanism value.

**Ex::-**

**101,102,103,105,106,108,109**

first() => 101

last () => 109

headSet(105) => 101,102,103

tailSet(105) => 105,106,108,109

subset(101,106) =>101,102,103,105

comparator() => null

**TreeSet**

**==========**

1. **Underlying datastructure :** is "balancedtree".

2. **insertion order preserved**: no(elements will stored based on some sorting

order)

3. **duplicated allowed :** no

4. **heterogenous elements :** no(Sorting can't be done if it is heterogenous)

allowed

5. **null insertion possible :** no

6. **interface implementation :** Serializable and Cloneable, but not

RandomAccess

**Constructor associated with TreeSet**

**============================**

**1. TreeSet t=new TreeSet();**

=> Elements added will be in default natrual sorting order

eg:: String => Alphabetical order

Number => Ascending order

**2. TreeSet t=new TreeSet(Comparator cmp);**

=> Elements will be added in TreeSet based on the Compartor written by

the programmer.

1. **TreeSet t=new TreeSet(Collection c);**
2. **TreeSet t=new TreeSet(SortedSet s);**

**eg#1.**

**import java.util.TreeSet;**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet();**

**ts.add("A");**

**ts.add("a");**

**ts.add("Z");**

**ts.add("B");**

**ts.add("L");**

**System.out.println(ts);//[A,B,L,Z,a]**

**ts.add(new Integer(10));//RE:ClassCastException**

**ts.add(null);//RE:NullPointerException**

**}**

**}**

* If we try to add heterogenous elements then it would result in

"ClassCastException".

* If we try to add null then it would result in "NullPointerException".

**Note:**

In the lower version below 1.8 in Treeset null as the first element was allowed, but in higer version even first element as null is also not allowed it would result in "NullPointerException".

**eg#2.**

**import java.util.TreeSet;**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet();**

**ts.add(new StringBuffer("A"));**

**ts.add(new StringBuffer("Z"));**

**ts.add(new StringBuffer("L"));**

**ts.add(new StringBuffer("B"));**

**System.out.println(ts);//**RE:ClassCastException

**}**

**}**

* When the elements are added into TreeSet based on natural sorting order the added object should be

1. Homogenous
2. It should implement Comparable interface

* If it fails to do jvm at the run time would throw an Exception called "ClassCastException".

**Note::**

String and all wrapper class has implemented an interface called "Comparable"

so only for String and Number type object default natural sorting order is

"Alphabetical and Ascending order".

**Comparable(I)**

=> It is a part of java.lang pacakge

=> It contains only one abstract method

public abstract int compareTo(Object obj)

**Behind the scenes**

**===============**

obj1.compareTo(obj2)

|=> returns -ve,iff obj1 has to come before obj2

|=> returns +ve,iff obj1 has to come after obj2

|=> returns 0,iff obj1 and obj2 are equal

**eg#1**

**System.out.println("A".compareTo("Z"));**

**System.out.println("L".compareTo("A"));**

**System.out.println("A".compareTo("A"));**

**System.out.println("A".compareTo(null));**

**eg#2**

**import java.util.TreeSet;**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet();**

**ts.add("K");**

**ts.add("Z");=> Z.compareTo("K)=> return +ve**

**ts.add("A");=> A.compareTo("K)=> return -ve**

**ts.add("A");=> A.compareTo("K")=>return -ve**

**A.compareTo("A")=> return 0**

**System.out.println(ts);//[A,K,Z]**

**ts.add(null);/**/RE:NullPointerException

**}**

**}**

If we are not satisified with default natural sorting internally done by jvm, then we customize the sorting order as per our requirement,to do so we need to take the help of "Comparator".

**Comparable**

=> It is a part of java.lang package

=> It has only one method called int compareTo(obj)

=> It is purely meant for natural sorting order

**Comparator**

=>It is a part of java.util package

=>It has 2 methods

public int compare(Object obj1,Object obj2)

public boolean equals(Object obj)

=>It is purely meant for customizing the sorting order

**Note::** while implementing an interface called Compartor, we can give body only

for onemethod called compare(Object obj1,Object obj2), where as for

equals() the body would come from Object class(inheritance at its best)

**eg#1**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**//body for compare method**

**}**

**}**

**Note::**

**public int compartor(Object obj1,Object obj2)**

|=> returns -ve,iff obj1 has to come before obj2

|=> returns +ve,iff obj1 has to come after obj2

|=> returns 0,iff obj1 and obj2 are equal

**Sort the objects in TreeSet**

**======================**

**import java.util.TreeSet;**

**import java.util.Comparator;**

**class MyComparator implements Comparator{**

**public int compare(Object obj1,Object obj2){**

**Integer I1=(Integer)obj1;**

**Integer I2=(Integer)obj2;**

**if (I1<I2)**

**return +1;**

**else if (I1>I2)**

**return -1;**

**else**

**return 0;**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet(new MyComparator());**

**ts.add(10);**

**ts.add(0);**

**ts.add(15);**

**ts.add(5);**

**ts.add(20);**

**ts.add(20);**

**System.out.println(ts);**

**}**

**}**

**Alternative ways of implementing Comparator interface**

**=========================================**

**class MyComparator implements Comparator{**

**public int compare(Object obj1,Object obj2){**

**Integer I1=(Integer)obj1;**

**Integer I2=(Integer)obj2;**

**return I1.compareTo(I2); //Ascending order**

**return -I1.compareTo(I2);//Descending order**

**return I2.compareTo(I1); //Descending order**

**return -I2.compareTo(I1);//Ascending Order**

**return +1;//Insertion order[0,5,10,15,20,20]**

**return -1;//revese of Insertion order[20,20,5,15,0,10]**

**return 0;//only first element will be added [10]**

**}**

**}**

**eg#2.** Sorting StringBuffer object based on String data present in it in reverse of

alphabetical order

**import java.util.TreeSet;**

**import java.util.Comparator;**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**String s1=obj1.toString();**

**String s2=obj2.toString();**

**return -s1.compareTo(s2);**

**or**

**return s2.compareTo(s1);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet(new MyComparator());**

**ts.add(new StringBuffer("A"));**

**ts.add(new StringBuffer("Z"));**

**ts.add(new StringBuffer("K"));**

**ts.add(new StringBuffer("L"));**

**System.out.println(ts);//[Z,K,L,A]**

**}**

**}**

**eg#3.**

Write a java code to sort the String objects in Descending order and store in TreeSet

**import java.util.TreeSet;**

**import java.util.Comparator;**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**String s1=(String)obj1;//underlying object is Still String so typecasting permitted**

**String s2=obj2.toString();**

**return s2.compareTo(s1);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet(new MyComparator());**

**ts.add("sachin");**

**ts.add("saurav");**

**ts.add("dhoni");**

**ts.add("dravid");**

**ts.add("zaheer");**

**ts.add("yuvi");**

**System.out.println(ts);**//[zaheer, yuvi, saurav, sachin,

dravid, dhoni]

**}**

**}**

Q. Write a java program to insert the String and StringBuffer object into TreeSet

where sorting order is in increasing length order.if 2 objects have same length

then consider their Alphabetical order

**sample::**

ts.add(new StringBuffer("A"));

ts.add(new StringBuffer("ABC"));

ts.add(new StringBuffer("AA"));

ts.add("XX");

ts.add("ABCE");

ts.add("A");

System.out.println(ts);//[A,AA,XX,ABC,ABCE]

**eg#1.**

**import java.util.TreeSet;**

**import java.util.Comparator;**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**String s1=obj1.toString();**

**String s2=obj2.toString();**

**int l1= s1.length();**

**int l2= s2.length();**

**if (l1<l2)**

**return -1;**

**else if(l1>l2)**

**return +1;**

**else**

**return s1.compareTo(s2);**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet(new MyComparator());**

**ts.add(new StringBuffer("A"));**

**ts.add(new StringBuffer("ABC"));**

**ts.add(new StringBuffer("AA"));**

**ts.add("XX");**

**ts.add("ABCE");**

**ts.add("A");**

**System.out.println(ts);//**[A,AA,XX,ABC,ABCE]

**}**

**}**

**Note::**

When to use Comparable and Comparator?

Comparable interface => If it is predefined class, then that class if it implements

Comparable then default natural sorting order would be

given.

If it is not given then we can't change the source code

to give the implemenation for Comparable interface,so

we prefer using "Comparator".

Comparator interface => If we are not happy with the default natural sorting order

for the objects we prefer using "Comparator".

Normally this would be used if the class is "Userdefined".

* While writing any userdefined class, if we want to give sorting order for an objects then we need to implement comparable interface for that class.
* If the user who uses our class if he is not happy with natural sorting order then we need to cusotmized using "Comparator" interface.

Q. Write a code to keep Employee Object in TreeSet

1. To promote default natural sorting,implement comparable interface and sort

the data based on property called "id".

1. Customize the natrual sorting order by implementing Comparator Interface

and sort the data based on property called "name".

**eg#1.**

**import java.util.TreeSet;**

**import java.lang.Comparable;**

**import java.util.Comparator;**

**class Employee implements Comparable{**

**int id;**

**String name;**

**Employee(int id,String name){**

**this.id=id;**

**this.name=name;**

**}**

**@Override**

**public String toString(){**

**return id+" ====> "+name;**

**}**

**@Override**

**public int compareTo(Object obj){**

**int id1=this.id;//adding object id**

**Employee e=(Employee)obj;**

**int id2=e.id;** //already added object id

//logic for ascending order sorting

**if(id1<id2)**

**return -1;**

**else if (id1>id2)**

**return +1;**

**else**

**return 0;**

**}**

**}**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**Employee e1=(Employee)obj1;**

**Employee e2=(Employee)obj2;**

**String s1=e1.name;**

**String s2=e2.name;**

**return s1.compareTo(s2);** //it is invoked on String Object

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet ts=new TreeSet();**

**ts.add(new Employee(10,"sachin"));**

**ts.add(new Employee(7,"ronaldo"));**

**ts.add(new Employee(18,"kohli"));**

**ts.add(new Employee(45,"rohith"));**

**ts.add(new Employee(9,"lara"));**

**ts.add(new Employee(14,"ponting"));**

**System.out.println(ts);**

**ts=new TreeSet(new MyComparator());**

**ts.add(new Employee(10,"sachin"));**

**ts.add(new Employee(7,"ronaldo"));**

**ts.add(new Employee(18,"kohli"));**

**ts.add(new Employee(45,"rohith"));**

**ts.add(new Employee(9,"lara"));**

**ts.add(new Employee(14,"ponting"));**

**System.out.println(ts);**

**}**

**}**

**Output**

**======**

NaturalSorting order

[7 ====> ronaldo, 9 ====> lara, 10 ====> sachin, 14 ====> ponting, 18 ====> kohli,45 ====> rohith]

CustomizedSorting order

[18 ====> kohli, 9 ====> lara, 14 ====> ponting, 45 ====> rohith, 7 ====> ronaldo, 10 ====> sachin]

**Difference b/w Comparable and Compartor interface?**

Comparable => meant for default natrual sorting order

Comparator => meant for custmoize sorting order

Comparable => java.lang package

Comparator => java.util package

Comparable => only one method int compareTo(Object obj)

Comparator => 2 method

int compare(Object obj1,Object obj2)

boolean equals(Object obj)

Comparable => Predefined internal implementation classes are String,Wrapper

class

Comparator => Predefined internal implementation classes are Collator and

RuleBaseCollator.

**Keypoints on implementation classes of Set**

**=================================**

HashSet => underlying datastructure is HashTable.

LinkedHashSet => underlying datastructure is LinkedList+HashTable.

TreeSet => underlying datastructure is balancedtree.

HashSet => insertion order not preserved

LinkedHashSet => insertion order is preserved

TreeSet => insertion order is not preserved

HashSet => heterogenous objects are allowed.

LinkedHashSet => heterogenous objects are allowed

TreeSet => heterogenous objects are not allowed(default nature)

HashSet => duplicate objects not allowed.

LinkedHashSet => duplicate objects not allowed.

TreeSet => duplicate objects not allowed.

HashSet => Sorting order not applicable.

LinkedHashSet => Sorting order not applicable.

TreeSet => Sorting order applicable.

HashSet => null acceptance allowed.

LinkedHashSet => null acceptance allowed.

TreeSet => null acceptance as first element allowed till jdk1.6

from 1.7V onward null acceptance is not allowed.

**Map**

**===**

1. It is not a child interface of Collection

2. To hold the Object as key-value pair we need to use Map.

3. Key can't be duplicated, Value can be duplicated.

4. Key and value both are treated as Objects

5. Every key and value pair is treated as "Entry".

**Common methods available in Map(I) for all the Map implementation class Object**

**==============================================================**

1. **Object put(Object Key,Object value)**

=> if the key already exists, then it would just add new value to that

key and old value will be returned to the user.

2. **Object putAll(Map m)**

3. **Object get(Object key)**

4. **Object remove(Object key)**

5. **boolean containsKey(Object key)**

6. **boolean containsValue(Object value)**

7. **boolean isEmpty()**

8. **int size()**

9. **void clear()**

-------------------------------

Collection view methods of Map

10.**Set keySet()**

11.**Collection values()**

12.**Set entrySet()**

**Entry**

**=====**

* Each key-value pair is called one Entry.
* Without existence of Map Object,Entry object won't exists.
* Interface Entry should be a part of Map interface.

**interface Map{**

**interface Entry{**

**Object getKey();**

**Object getValue();**

**Object setValue(Object value);**

**}**

**}**

**HashMap**

**=======**

* **Underlying datastructure:** hashtable.
* **heterogenous elements :** yes.
* **duplicated allowed :** keys not allowed,but values can be duplicated.
* **insertion order :** not preserved because of Hashing technique(hashCode

value of keys)

* **null insertion :** allowed for keys (only once),but for values (any no).

**Constructor**

**===========**

1. **HashMap hm=new HashMap();**

//Default capacity => 16

// load factor => 0.75

2. **HashMap hm=new HashMap(int initialCapacity);**

3. **HashMap hm=new HashMap(int initalCapacity,int fillratio);**

4. **HashMap hm=new HashMap(Map m);**

**eg#1.**

**import java.util.HashMap;**

**import java.util.Set;**

**import java.util.Collection;**

**import java.util.Iterator;**

**import java.util.Map;**

**public class TestApp {**

**public static void main(String[] args){**

**HashMap hm= new HashMap();**

**hm.put(10,"sachin");**

**hm.put(7,"dhoni");**

**hm.put(18,"kohli");**

**hm.put(19,"dravid");**

**System.out.println(hm);** //{18=kohli, 19=dravid, 7=dhoni,

10=sachin}

**System.out.println(hm.put(10,"messi"));**//sachin

**System.out.println(hm);** //{18=kohli, 19=dravid, 7=dhoni,

10=messi}

**Set s=hm.keySet();**

**System.out.println(s);** //[18,19,7,10]

**Collection c = hm.values();**

**System.out.println(c);** //[kohli,dravid,dhoni,messi]

**Set s1=hm.entrySet();**

**System.out.println(s1);**//[18=kohli,19=dravid,7=dhoni,10=sachin]

**Iterator itr=s1.iterator();**

**while(itr.hasNext()){**

**Map.Entry entry=(Map.Entry)itr.next();**

**System.out.println(entry.getKey() +"----> "+**

**entry.getValue());**

**if (entry.getValue().equals("messi"))**

**entry.setValue("sachin");**

**}**

**System.out.println(hm);** //{18=kohli, 19=dravid, 7=dhoni,

10=sachin}

**}**

**}**

**Difference b/w HashMap and Hashtable?**

HashMap => methods are non-synchronized.

Hashtable=> methods are synchronized

HashMap => At a time multiple threads can act on a object, so it is not

ThreadSafe.

Hashtable => At a time only one thread can act on a object, so it is ThreadSafe.

HashMap => performance is high.

Hashtable => performance is low.

HashMap => introduced in 1.2V not a legacy class.

Hashtable => introduced in 1.0v is a legacy class.

HashMap => null insertion as a key or value is permitted.

Hashtable => null insetion as a key or value is not permited,it would result in NullPointerException.

**Note::** To get Synchronized version of HashMap, we use a method called

sycnchronizedMap() using Collections class(utility class).

**LinkedHashMap**

**=============**

* **Underlying datastructure:** linkedlist + hashtable.
* **heterogenous elements :** yes.
* **duplicated allowed**  **:** keys not allowed,but values can be duplicated.
* **insertion order :** not preserved because of Hashing technique(hashCode

value of keys)

* **null insertion :** allowed for keys (only once),but for values (any no).

if we replace HashMap with LinkedHashMap then the output of the above pgm would be

{10=sachin, 7=dhoni, 18=kohli, 19=dravid}

sachin

{10=messi, 7=dhoni, 18=kohli, 19=dravid}

[10, 7, 18, 19]

[messi, dhoni, kohli, dravid]

[10=messi, 7=dhoni, 18=kohli, 19=dravid]

10----> messi

7----> dhoni

18----> kohli

19----> dravid

{10=sachin, 7=dhoni, 18=kohli, 19=dravid}

**Note::** HashMap and LinkedHashMap best suited when we develop cache based

application where duplicates should not be allowed,but focus only on the

object and its insertion order.

**IdentityHashMap(1.4v)**

**=====================**

It is almost similar to HashMap only but with only one difference.

1. HashMap uses equals() to compare the duplication of keys which is meant

for content comparison.

1. IdentityHashMap uses == to check for the duplication of keys which is meant for reference comparison.

**eg#1.**

**import java.util.HashMap;**

**public class TestApp {**

**public static void main(String[] args){**

**HashMap hm= new HashMap();**

**Integer i1=new Integer(10);**

**Integer i2=new Integer(10);**

**hm.put(i1,"sachin");**

**hm.put(i2,"messi");**

**System.out.println(hm);** //{10=messi}

**}**

**}**

If the above code is replace with IdentityHashMap then the output would be {10=sachin, 10=messi}.

**WeakHashMap**

**===========**

WeakHashMap is almost same as HashMap,but with the following difference

1. HashMap would dominate GarbageCollector, that is if Object does not have

any reference still it is not eligible for garabage collection as it is associated with HashMap.

1. WeakHashMap can't dominate GarbageCollection, that is if Object does not

have any reference then it is eligible for Garbage Collection even though it is associated with WeakHashMap.

**eg#1.**

**import java.util.HashMap;**

**public class TestApp {**

**public static void main(String[] args)throws Exception{**

**HashMap hm= new HashMap();**

**Temp t= new Temp();**

**hm.put(t,"nitin");**

**System.out.println(hm);//{temp=nitin}**

**t=null;**

**System.gc();**

**Thread.sleep(5000);//5sec**

**System.out.println(hm);//**{temp=nitin}

**}**

**}**

**class Temp{**

**@Override**

**public String toString(){**

**return "temp";**

**}**

**@Override**

**public void finalize(){**

**System.out.println("garbage collector is cleaning the object");**

**}**

**}**

If we replace HashMap with WeakHashMap, then the output would be

{temp=nitin}

garbage collector is cleaning the object

{}

**SortedMap**

**========**

* It is a child interface of a Map.
* If we want to represent a group of key-value pair based on some sorting order then we need to use "SortedMap".

**SortedMap defines the following specific methods**

**====================================**

**1. Object firstKey()**

**2. Object lastKey()**

**3. SortedMap headMap(Object key)**

**4. SortedMap tailMap(Object key)**

**5. SortedMap subMap(Object key1,Object key2)**

**6. Compartor compartor()**

**TreeMap**

**=======**

* **Underlying datastructure is :** red-black tree.
* **duplicate allowed :** keys not allowed,values are allowed.
* **insertion order :** not preserved,but it will stored based on some

sorting order.

* **heterogenous objects :** w.r.t to keys not allowed,it would result in

ClassCast Exception value Object can be

heterogenous.

* **null values :** it would result in nullpointer exception after jdk1.7

before jdk1.6 only as the first key element it was

allowed.

Since in TreeMap,Sorting will happen based on keys we need to refer to

1. Comparable(I)=> default natural sorting order

2. Comparator(I)=> customized sorting order

**Constructor in TreeMap**

**======================**

**1. TreeMap t=new TreeMap();**

**2. TreeMap t=new TreeMap(Comparator c);**

**3. TreeMap t=new TreeMap(SortedMap s);**

**4. TreeMap t=new TreeMap(Map m);**

**eg#1.**

**import java.util.TreeMap;**

**public class TestApp {**

**public static void main(String[] args){**

**TreeMap tm=new TreeMap();**

**tm.put("ZZZ",10);**

**tm.put("AAA",100);**

**tm.put("LLL",null);**

**tm.put("CCC",10);**

**tm.put("ZZZ",150);**

**System.out.println(tm);** //{AAA=100, CCC=10, LLL=null, ZZZ=150}

**tm.put(null,10);** //RE:NullPointerException

**tm.put(10,10);** //RE:ClassCastException

**}**

**}**

**eg#2.**

**import java.util.TreeMap;**

**import java.util.Comparator;**

**public class TestApp {**

**public static void main(String[] args){**

**TreeMap tm=new TreeMap(new MyComparator());**

**tm.put("ZZZ",10);**

**tm.put("AAA",20);**

**tm.put("LLL",40);**

**tm.put("CCC",30);**

**tm.put("XXX",150);**

**System.out.println(tm);**//{ZZZ=10,XXX=150,LLL=40,CCC=30,AAA=20}

**}**

**}**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**String s1=(String)obj1;**

**String s2=obj2.toString();**

**return s2.compareTo(s1);**

**}**

**}**

**Legacy classes**

**==============**

Hashtable(1.0v)

* **Underlying datastructure is :** hashtable only.
* **duplicate allowed :** keys not allowed,values are allowed.
* **insertion order :** not preserved becoz it is based on hashcode of key

Object.

* **heterogenous objects :** allowed for both keys and values
* **null values :** not allowed for both keys and values, it would result in

NullPointerException.

**Constructor**

**==========**

1. **Hashtable hm=new Hashtable();**

//Default capacity =>11

//Fill ratio =>0.75

2. **Hashtable hm=new Hashtable(int initialCapacity);**

3. **Hashtable hm=new Hashtable(int capacity,int fillratio);**

4. **Hashtable hm=new Hashtable(Map m);**

**eg#1.**

**import java.util.Hashtable;**

**public class TestApp {**

**public static void main(String[] args){**

**Hashtable ht= new Hashtable();**

**ht.put(new Temp(5),"A");**

**ht.put(new Temp(2),"B");**

**ht.put(new Temp(6),"C");**

**ht.put(new Temp(15),"D");**

**ht.put(new Temp(23),"E");**

**ht.put(new Temp(16),"F");**

**System.out.println(ht);** //{6=C, 16=F, 5=A, 15=D, 2=B, 23=E}

**}**

**}**

**class Temp{**

**int i;**

**Temp(int i){**

**this.i=i;**

**}**

**@Override**

**public int hashCode(){**

**return i;**

**}**

**@Override**

**public String toString(){**

**return i+"";**

**}**

**}**

**Scenario2:**

if hashcode() is changed to

public int hashCode(){

return i%9;

}

then output would be {16=F, 15=D, 6=C, 23=E, 5=A, 2=B}

**Scenario3:**

if hashcode() is changed to

public int hashCode(){

return i%9;

}

but hashtable is created using the following constructor

Hashtable hm = new Hashtable(25);

**Properties**

**==========**

1. It is a child class of Hashtable

2. In a application, if a data changes frequently then those variables should not

be hardcoded because as the value changes to the varaible we need to

a. recompile

b. rebuild

c. redeploy the application which would have huge impact at buisness level.

1. To avoid this we need to hardcode this values at the properties file and read those values from the properties file to the java application so recompilation and rebuilding won't happen it is just redeployment.

4. To work with properties file at the java end we need to use "Properties" Object.

**Constructor**

**==========**

**Properties properties=new Properites()**

**Note::** While creating a properties the key and value would always be in String

format only.

String url="";//java level

application.properties

url=jdbc:mysql:///abc

username=root

password=root123

**Methods associated with Properties Object**

**=================================**

1. public String getProperty(String keyName)

2. public void setProperty(String keyName,String value)

3. public void load(InputStream is)

4. public void store(OutputStream os,String data)

5. public Enumeration propertyNames()

**eg#1.**

**import java.util.Properties;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.util.Enumeration;**

**public class TestApp {**

**public static void main(String[] args) throws Exception{**

//Step1: Create a properties object to hold Properties file

**Properties p=new Properties();**

//Step2: Create a FileInputStream to bring the properties file into Properties Object

**FileInputStream fis =new FileInputStream("application.properties");**

**p.load(fis);**

**System.out.println(p);**//{url=jdbc:mysql:///abc,username=root,password=root123}

**System.out.println("The url is "+p.getProperty("url"));**

**System.out.println("The username is "+p.getProperty("username"));**

**System.out.println("The password is "+p.getProperty("password"));**

**Enumeration e=p.propertyNames();**

**while(e.hasMoreElements()){**

**String data=(String)e.nextElement();**

**System.out.println(data);**

**}**

**FileOutputStream fos= new FileOutputStream("application.properties");**

**p.store(fos,"Thanks for giving the db infromation....");**

**}**

**}**

**application.properties**

**=======================**

#Thanks for giving the db infromation....

#Thu Feb 03 10:59:51 IST 2022

password=root123

url=jdbc:oracle:thin:@localhost:1521:XE

username=System

**eg#1.**

**import java.util.Properties;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.util.Enumeration;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**public class TestApp {**

**public static void main(String[] args) throws Exception{**

//Step1: Create a properties object to hold Properties file

**Properties p=new Properties();**

//Step2: Create a FileInputStream to bring the properties file into Properties Object

**FileInputStream fis =new FileInputStream("application.properties");**

**p.load(fis);**

**String url=p.getProperty("url");**

**String username=p.getProperty("username");**

**String password=p.getProperty("password");**

**Connection connection=DriverManager.getConnection(url,username,password);**

**System.out.println("Connection Established to "+url);**

**}**

**}**

Just by changing the properties file,same code can be used to communicate with multiple databases

through which we can say java slogan "WORA".

**Queue(I)**

**========**

1. It is a child interface of Collection

2. If we want to represent group of individual objects Prior to Processing then

we should go for Queue.

3. From 1.5 version LinkedList implements Queue

4. Usually Queue follows FIFO order,but based on our needs we can implement

our own Priorities.

eg:PriorityQueue

1. LinkedListBased Implementation Queue also follows FirstInFirstOut Order

only(FIFO).

eg: If our application wants to send some message,mail to the clients before

sending it will collect in one datastructure that datastructure should support

of keeping the data in same order the data is been sent so in the same order of

storage the processing can happen.

In this scenario the best suited datastructrue is "Queue".

**Methods**

**=======**

boolean offer(Object obj)

=> To add an Object into the Queue

Object peek()

=> Returns the first element of the Queue

=> If Queue is empty it returns null

Object element()

=> Returns the first element of the Queue

=> If Queue is empty it throws an Exception called

"NoSuchElementException".

Object poll()

=> It removes and returns the first element of the Queue

=> If Queue is empty it returns null

Object remove()

=> It removes and returns the first element of the Queue

=> If Queue is empty it returns null

**PriorityQueue**

**=============**

* **Underlying DataStructure**=> Queue
* **Insertion order** => Based on the priority the elements will be inserted
* **order** => default natrual sorting order
* **customization** => possible,but should implement Comparator.
* **Duplicates**  => not allowed
* **Heterogenous elements** => if we are dependant on natural sorting order

then not allowed

if it is based on customisation then it can be allowed.

* **null elements** => not allowed.

**Constructors**

**============**

**PriorityQueue p=new PriorityQueue()**

//Default Capacity => 11

**PriorityQueue p=new PriorityQueue(int capacity)**

**PriorityQueue p=new PriorityQueue(int capacity, Comparator c)**

**PriorityQueue p=new PriorityQueue(SortedSet s)**

**PriorityQueue p=new PriorityQueue(Collection c)**

**eg#1.**

**import java.util.PriorityQueue;**

**public class TestApp {**

**public static void main(String[] args){**

**PriorityQueue<Integer> p = new PriorityQueue<Integer>();**

**//System.out.println(p.poll());**

**//System.out.println(p.remove());**

**for (int i=0; i<=10;i++ ){**

**p.offer(i);**

**}**

**System.out.println(p);//[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

**System.out.println(p.poll());//0**

**System.out.println(p);//[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

**}**

**}**

**Note::** Some operating System, wont give support for PriorityQueue.

To get the support we need to install batch files from the Service

provider(Microsoft).

**eg#2.**

**import java.util.PriorityQueue;**

**public class TestApp {**

**public static void main(String[] args)**

**{**

**PriorityQueue<String> q = new PriorityQueue<String>(15,new MyComparator());**

**q.offer("Z");**

**q.offer("A");**

**q.offer("L");**

**q.offer("X");**

**System.out.println(q);//[Z, X, L, A]**

**}**

**}**

**class MyComparator implements java.util.Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**String s1=(String)obj1;**

**String s2=obj2.toString();**

**return -s1.compareTo(s2);**

**}**

**}**

**Enhancement w.r.t 1.6 version**

**===========================**

1. **NavigableSet**

a.It is a child interface of SortedSet.

b.This interface provides methods which are meant for navigation purpose.

**Specific methods associated with NavigableSet**

**===================================**

1. **floor(Object obj)**

=> It returns the highest element <=e

2. **lower(Object obj)**

=> It returns the highest element <e

3. **ceiling(Object obj)**

=> It returns the lowest element >=e

4. **higher(Object obj)**

=> It returns the lowest element >e

5. **pollFirst()**

=> It removes and returns the first element of the Set.

6. **pollLast()**

=> It removes and returns the last element of the Set.

7. **descendingSet()**

=> It returns the navigable set in reverse order.

**eg#1.**

**import java.util.TreeSet;**

**public class TestApp {**

**public static void main(String[] args){**

**TreeSet<Integer> ts = new TreeSet<Integer>();**

**ts.add(1000);**

**ts.add(2000);**

**ts.add(3000);**

**ts.add(4000);**

**ts.add(5000);**

**System.out.println(ts);** //[1000, 2000, 3000, 4000, 5000]

**System.out.println(ts.floor(3000));** //3000[highest element <=3000]

**System.out.println(ts.lower(3000));** //2000[highest element <3000]

**System.out.println(ts.ceiling(2000));** //2000[lowest element >=2000]

**System.out.println(ts.higher(2000));** //3000 [lowest element >2000]

**System.out.println(ts.pollFirst());** //1000[remove and return first element]

**System.out.println(ts.pollLast());** //5000 [remove and reutrn last element]

**System.out.println(ts);** //[2000,3000,4000]

**System.out.println(ts.descendingSet());** //[4000,3000,2000]

**}**

**}**

**NavigableMap**

**============**

1. It is a child interface of SortedMap

**Specific methods associated with NavigableMap**

**=====================================**

**1. floorKey(Object obj)**

**2. lowerKey(Object obj)**

**3. ceilingKey(Object obj)**

**4. higherKey(Object obj)**

**5. pollFirstEntry()**

**6. pollLastEntry()**

**7. descendingMap()**

**import java.util.TreeMap;**

**public class TestApp {**

**public static void main(String[] args){**

**TreeMap<String,String> tm=new TreeMap<String,String>();**

**tm.put("b","Banana");**

**tm.put("c","Cat");**

**tm.put("a","Apple");**

**tm.put("d","Dog");**

**tm.put("g","Gun");**

**System.out.println(tm);** //{a=Apple, b=Banana, c=Cat, d=Dog, g=Gun}

**System.out.println(tm.ceilingKey("c"));//c**

**System.out.println(tm.higherKey("e"));//g**

**System.out.println(tm.floorKey("e"));//d**

**System.out.println(tm.lowerKey("e"));//d**

**System.out.println(tm.pollFirstEntry());//{a=Apple}**

**System.out.println(tm.pollLastEntry());//{g=Gun}**

**System.out.println(tm.descendingMap());//{d=Dog,c=Cat,b=Banana}**

**}**

**}**

**UtilityClass**

**============**

**1. Collections**

**2. Arrays**

**Collections**

**===========**

It is a class which is a part of java.util.Collections

It defines several methods which are meant for Collection Objects.

**Sorting the elements of the List**

**=========================**

**1**. Collection class defines the method to perform Sorting the elements of the List

public static void sort(List l);

To sort the elements in default natural sorting order the elements should be

a. homogeneous.

b. comparable interface.

otherwise it would result in "ClassCastException/NullPointerException".

**eg#1.**

**import java.util.ArrayList;**

**import java.util.Collections;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al= new ArrayList();**

**al.add("Z");**

**al.add("A");**

**al.add("K");**

**al.add("N");**

**//al.add(null);**

**//al.add(new Integer(10));**

**System.out.println("Before sorting :: "+ al);** //[Z,A,K,N,10]

**Collections.sort(al);//RE: ClassCastException**

**System.out.println("Before sorting :: " + al);** //[A,K,N,Z]

**}**

**}**

**2.** sort() present in Collections class is a Overloaded method.

**3.** If we want to sort the Objects based on our own sorting order then we need to

use public static void sort(List l,Comparator c)

|=> calls compare() on every Object

**Rule:**

1. The elements need not be homogeneous.

2. The eleemnts need not implements Comparable.

3. The element should not be null otherwise "NullPointerException".

**eg#2.**

**import java.util.ArrayList;**

**import java.util.Collections;**

**import java.util.Comparator;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al= new ArrayList();**

**al.add("Z");**

**al.add("A");**

**al.add("K");**

**al.add("N");**

**al.add(new StringBuffer("a"));**

**System.out.println("Before sorting :: " + al);** //[Z,A,K,N,a]

**Collections.sort(al,new MyComparator());**

**System.out.println("Before sorting :: " + al);** //[a,Z,N,K,A]

**}**

**}**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**String s1=obj1.toString();**

**String s2=obj2.toString();**

**return s2.compareTo(s1);**

**}**

**}**

**2. Searching for an element in the list**

**=============================**

1. It is used for searching the elements in the list

2. It internally uses binaryserach algorithm to search for a particular key.

3. Before calling binarySerach(), we need to make sure the elements in the list is

in "SortedOrder" otherwise it would result in "Unpredictable result".

public static int binarySearch(List l,Object o);

**eg#1.**

**import java.util.ArrayList;**

**import java.util.Collections;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al= new ArrayList();**

**al.add("Z");**

**al.add("A");**

**al.add("M");**

**al.add("K");**

**al.add("a");**

**Collections.sort(al);**

**System.out.println(al);**//[A,K,M,Z,a]

**System.out.println(Collections.binarySearch(al,"Z"));**//3

**System.out.println(Collections.binarySearch(al,"b"));**//-6

**System.out.println(Collections.binarySearch(al,"L"));**//-3

**}**

**}**

If it is succesfull in searching, it returns the index no

If it is not succesfull in searching,it return the insertion point no.

Date: 08/02/2022

Topic: CollectionFramework

==========================

**9 Interfaces are a part of CollectionFramework**

**===================================**

1. Collection(I)

2. List(I)

3. Set(I)

4. SortedSet(I)

5. NavigableSet(I)

6. Queue(I)

7. Map(I)

8. SortedMap(I)

9. NavigableMap(I)

**Curosrs in Collection Framework**

**===============================**

1. Enumeration(I)

2. Iterator(I)

3. ListIterator(I)

**Utility Classes**

**===============**

1. Arrays

2. Collections

**Based on Sorting to happen**

**==========================**

1. Comparable(I)=> default natural sorting order

2. Comparator(I)=> customized sorting order

**Collections Utility class**

**========================**

1. Sorting

2. Searching

3. reversing

**Seraching**

=> It internally uses binary search algorithm to search for a key element

=> If key element is found it would return the index of that element otherwise

it will return the insertion point of that element in the list.

**eg#1.**

**import java.util.ArrayList;**

**import java.util.Collections;**

**import java.util.Comparator;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al= new ArrayList();**

**al.add(15);**

**al.add(0);**

**al.add(20);**

**al.add(10);**

**al.add(5);**

**Collections.sort(al,new MyComparator());**

**System.out.println(al);//[20,15,10,5,0]**

**System.out.println(Collections.binarySearch(al,5,new MyComparator()));**//3

**System.out.println(Collections.binarySearch(al,13,new MyComparator()));**//-3

**System.out.println(Collections.binarySearch(al,5));** //-1[unexpected result]

**}**

**}**

**class MyComparator implements Comparator{**

**@Override**

**public int compare(Object obj1,Object obj2){**

**Integer i1=(Integer)obj1;**

**Integer i2=(Integer)obj2;**

**return i2.compareTo(i1);** //reverse of natural sorting order

**}**

**}**

**Conclusion**

**==========**

1. Internally these methods uses binary search algorithm.

2. on succesfull search it returns the index, otherwise it return the insertion point

of that particular key.

3. Before calling binarySearch() compulsorily the object must be sorted othewise

it would result in UnExpected result.

4. If the List object is sorted using Comparator,at the time of search operation

compulsorily we need to give Comparator Object otherwise also it would

result in "UnExpected" result.

**BinarySerachRange of Searching the element is** => -(n+1) to n-1

**succesfull searching of index range is** => o to n-1

**unsuccesfull searching of index range is** => -(n+1) to -1

=================================================

**Total searching of index range i**s => -(n+1) to n-1

**Reversing**

**========**

This mechanism is basically used to revers the order of elements

public void reverse(List l)

**import java.util.ArrayList;**

**import java.util.Collections;**

**public class TestApp {**

**public static void main(String[] args){**

**ArrayList al= new ArrayList();**

**al.add(15);**

**al.add(0);**

**al.add(20);**

**al.add(10);**

**al.add(5);**

**System.out.println("Before reversing :: "+al);** //[15,0,20,10,5]

**Collections.reverse(al);**

**System.out.println("After reversing :: "+al);** //[]

**}**

**}**

**2. reverseOrder()**

**=============**

It is used to get the reversed Comparator Object

**Comparator c2=Collections.reverseOrder(Comparator c1);**

| |

Descending order Ascending order.

**Arrays**

**=====**

1. **Sorting the elements of the Array**

**public void sort(primitive[] p)**

It supports default natural sorting order of elements only primitive

type.

**public void sort(Object[] p)**

It support default natural sorting order of element of Objects.

**public void sort(Object[] p,Comparator c);**

It support Customized sorting order of element of Objects.

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.Comparator;**

**public class TestApp {**

**public static void main(String[] args){**

**int[] arr={10,5,20,11,6};**

**System.out.println("Array elements before sorting");**

**for (int i:arr )**

**System.out.print(i+"\t");** //10 5 20 11 6

**Arrays.sort(arr);**

**System.out.println("\nArray elements after sorting");**

**for (int i:arr )**

**System.out.print(i+"\t");** //5 6 10 11 20

**System.out.println();**

**String[] sarr={"Z","A","B"};**

**System.out.println("\nArray elements before sorting");**

**for (String i:sarr )**

**System.out.print(i+"\t");** // Z A B

**Arrays.sort(sarr);**

**System.out.println("\nArray elements after sorting");**

**for (String i:sarr )**

**System.out.print(i+"\t");** // A B Z

**System.out.println();**

**System.out.println("\nCustomization of Sorting the elements");**

**Arrays.sort(sarr,new MyComparator());**

**System.out.println("\nArray elements after sorting");**

**for (String i:sarr )**

**System.out.print(i+"\t");** // Z B A

**System.out.println();**

**}**

**}**

**class MyComparator implements Comparator{**

**public int compare(Object obj1,Object obj2){**

**String s1= obj1.toString();**

**String s2= obj2.toString();**

**return s2.compareTo(s1);**

**}**

**}**

**2. Searching for an element in the Array**

1. It internally uses binary search algorithm to identify the key element present

in Array.

2. **public int binarySearch(primitive[] p,primitive key)**

**public int binarySearch(Object[] obj,Object obj)**

**public int binarySearch(Object[] obj,Object obj,Compartor c)**

3. All the above methods of binarySearch would work in the same way as that

of Collections class.

**eg#1.**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.Comparator;**

**public class TestApp {**

**public static void main(String[] args){**

**int[] arr={10,5,20,11,6};**

**Arrays.sort(arr);//5 6 10 11 20**

**System.out.println(Arrays.binarySearch(arr,5));//0**

**System.out.println(Arrays.binarySearch(arr,14));//-5**

**System.out.println();**

**String[] sarr={"A","Z","B"};**

**Arrays.sort(sarr);//A B Z**

**System.out.println(Arrays.binarySearch(sarr,"B"));//1**

**System.out.println(Arrays.binarySearch(sarr,"a"));//-4**

**System.out.println();**

**Arrays.sort(sarr,new MyComparator());//Z B A**

**System.out.println(Arrays.binarySearch(sarr,"A",new MyComparator()));//2**

**System.out.println(Arrays.binarySearch(sarr,"a",new MyComparator()));//-1**

**System.out.println(Arrays.binarySearch(sarr,"Z"));** //Unexpected result

**}**

**}**

**class MyComparator implements Comparator{**

**public int compare(Object obj1,Object obj2){**

**String s1= obj1.toString();**

**String s2= obj2.toString();**

**return s2.compareTo(s1);**

**}**

**}**

**Another commonly used utility method**

**=====================================**

1. toString()

2. asList()

* To convert any Collection object to an Array we have method called

public Object[] toArray()

* To convert Array into any Collection Object we have method called

public static List asList(Object[] obj)

**KeyPoints while working with asList()**

**====================================**

1. Using asList(), strictly speaking we are not creating any new List Object rather the same Array Object will be viewed as a List.
2. By using List reference, if we modify the Object,the result would be reflected

in "ArrayObject" also.

1. If we try to use List Object reference, and if we increase or decrease the size

of the list it would result in "UnsupportedOperation" becoz still the underlying datastructure is an Array Only and for an Array we can't increase or decrease the size.

1. If we try to use List Object reference, and if we update the list with

heterogenous elements it would result in "ArrayStoreException" becoz still the underlying datastructure is an Array Only and an Array can't hold heterogenous Objects.

**eg#1.**

**import java.util.Arrays;**

**import java.util.List;**

**public class TestApp {**

**public static void main(String[] args){**

**String[] s= {"A","Z","B"};**

**List l= Arrays.asList(s);**

**System.out.println(l);//AZB**

**l.add("sachin");//UnsupportedOperationException**

**l.remove("Z");//UnsupportedOperationException**

**l.set(1,"X");**

**for(String data:s)**

**System.out.print(data);//AXB**

**l.set(1,new Integer(10));//ArrayStoreException**

**}**

**}**

**Need of Generics**

**=\*=\*=\*=\*=\*=\*=\*=**

**Case1::**

Arrays are always type safe,that is we can give guarantee for the type of

elements stored inside the Array.

**String[] s=new String[1000];**

**s[0]="sachin";**

**s[1]="dhoni";**

**s[2]="yuvi";**

**s[3]=new Integer(10)**;//CE: incompatible type found :java.lang.Integer

requried :java.lang.String

Arrays always provides guarantee about the type of element what is been added,so we say Arrays are always "TypeSafe".

**ArrayList al=new ArrayList();**

**al.add("sachin");**

**al.add("dhoni");**

**al.add("yuvi");**

**al.add(new Integer(10));**

**String name1=(String)al.get(0);**

**String name2=(String)al.get(1);**

**String name3=(String)al.get(2);**

**String name4=(String)al.get(3);**//RE: ClassCastException : java.lang.Integer

can't be converted to java.lang.String.

**Note:** If we work with Collection then their is no gurantee of TypeSafety.

**Case2:**

In Arrays we don't want to perform Type casting at the retreival operation.

eg#1.

**String[] s=new String[1000];**

**s[0]="sachin";**

**s[1]="dhoni";**

**s[2]="yuvi";**

**String name=s[0];**

**System.out.println(name)**;//At the time of retrieval typecasting is not required.

But in case of Collection, at the time of retrieval typecasting is mandatory as the return type of the get() is Object.

**ArrayList al=new ArrayList();**

**al.add("sachin");**

**al.add("dhoni");**

**al.add("yuvi");**

**al.add(new Integer(10));**

**String name1=l.get(0);//**CE: => incompatible type required :java.lang.Object

found :java.lang.String

**String name2=(String)l.get(3);** => RE : ClassCastException

required :java.lang.Integer

found: java.lang.String

To resolve the above mention problem SUNMS people introduced a new concept called Generics,to make the usage of Collection friendly for the developers.

**Collection along with Generics would resolve the problem of**

a. It avoids typesafety.

b. It avoid typeCasting problem

**Syntax of Using Collection along with Generic**

**=============================================**

**BaseType<paramtertype> al =new BaseType<parametertype>();**

**eg: ArrayList<String> al =new ArrayList<String>();**

**al.add("sachin");**

**al.add("dhoni");**

**al.add("yuvi");**

**al.add(new Integer(10)**);//CE: symbol: add(int) not found

**Date: 14/02/2022**

**Topic: Generics**

**==========================**

**Need of Generics is**

a. To promote type safety.

b. To avoid TypeCasting at the run time.

**GeneralSyntax**

**=============**

**basetype<parametertype> ref=new basetype<parametertype>();**

**eg::**

**ArrayList<String> al =new ArrayList<String>();**

|=> In this list object we can keep only String type of Objects.

**Conclusion**

**==========**

1.

**List<String> al =new ArrayList<String>();**

**Collection<String> al =new ArrayList<String>();**

**List<Object> al =new ArrayList<String>()**;//invalid

**Polymporhism** => 1:M(collection child class object reference in parent type and behaviour getting invoked based on runtime object).

Polymporphism concept is applicable only for basetype it is not applicable for parameter type.

2.

**List<int> al =new ArrayList<int>()**;//invalid

Generic type should always be of referenc type like classname/interface

name it can't be of primitive type,if we take it would result in

"CompileTimeError".

**ArrayListVersion till jdk1.4**

**=============================**

**class ArrayList{**

**add(Object o)**

**Object get(int index)**

**}**

=> add method can take any parameter so we can't bring type safety.

=> get method return type is Object, so compulsorily we need to use type casting

to get the value.

**From JDK1.5 +**

**=============**

|=> Type Parameter

**class ArrayList<T>{**

**add(T t)**

**T get(int index)**

**}**

=> Based on our requirement T will be replaced with specific type.

=> get method return type is of specific type,so type casting is not required.

**example**

**======**

ArrayList class will be changed to

**class ArrayList<String>{**

**add(String t)**

**String get(int index)**

**}**

**ArrayList<String> al =new ArrayList<String>();**

**al.add("sachin");**

**al.add(new Integer(10));** //CE: incompatible type

required:java.lang.String

found :java.lang.Integer

**ArrayList<String> al =new ArrayList<String>();**

**al.add("sachin");**

**al.add("kohli");**

**al.add("saurav");**

**String name= al.get(0);**//Type casting not required, so it would avoid ClassCastException.

**Usage of Generics in Development**

**================================**

This Generics concpet is not only applicable for predefined API,it is also

applicable for user defined classes also.

eg#1.

|=> Type parameter

**class Account<T>{**

**}**

**Account<Gold> acc=new Account<Gold>();**

**Account<Silver> silverAccount=new Account<Silver>();**

eg#1.

**class Gen<T>{**

**T obj;**

**Gen(T obj){**

**this.obj=obj;**

**}**

**public void show(){**

**System.out.println("The type of object is :: "+obj.getClass().getName());**

**}**

**public T getObject(){**

**return obj;**

**}**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Gen<String> g1= new Gen<String>("sachin");**

**System.out.println(g1.getObject());**

**g1.show();**

**System.out.println();**

**Gen<Integer> g2= new Gen<Integer>(10);**

**System.out.println(g2.getObject());**

**g2.show();**

**System.out.println();**

**Gen<Double> g3= new Gen<Double>(10.5);**

**System.out.println(g3.getObject());**

**g3.show();**

**}**

**}**

**output**

sachin

The type of object is :: java.lang.String

10

The type of object is :: java.lang.Integer

10.5

The type of object is :: java.lang.Double

**BoundedTypes in Generics**

**=========================**

As noticed below, we can replace the Type parameter <T> with any type like

Integer,String,Double,.. and so on.

so we say T is "unbounded type".

|=> Type parameter

**class Test<T>{**

**}**

**Test<Integer> t1=new Test<Integer>();**

**Test<String> t2=new Test<String>();**

**To make TypeParameter as a bounded type we need to refer to the following syntax**

**======================================================**

**class Test < T extends X>**

|=>It means we can pass X, or its child classes only, other type

if we pass, it would result in "CompileTimeError".

**eg#1.**

**class Test<T extends Number>{**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Test<Integer> t1=new Test<Integer>();**

**Test<String> t2=new Test<String>()**;//CE: String not bounded to

Type-variable

**}**

**}**

**eg#2.**

**class Test<T extends Thread>{**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Test<Thread> t1=new Test<Thread>();**

**Test<String> t2=new Test<String>()**;//String is not with in

bounded type variable.

**}**

**}**

**eg#3.**

**class Test<T extends Runnable>{**

**}**

**public class TestApp {**

**public static void main(String[] args){**

**Test<Thread> t1=new Test<Thread>();**

**Test<String> t2=new Test<String>()**;//String is not with in

bounded type variable.

**}**

**}**

**Syntax supported by Generics**

**============================**

**class Test<T extends Number>{}** //valid

**class Test<T extends Thread>{}** //valid

**class Test<T extends Runnable>{}**//valid

class Test<T implements Runnable>{}//invalid

class Test<T super String>{}//invalid

**Note::** In generics to define bounded type we can't use implements and super

keyword.

Date: 15/02/2022

Topic: Generics

==========================

**Need of Generics is**

a. To promote type safety.

b. To avoid TypeCasting problems at the run time.

**Bounded Type in combination**

**===========================**

eg#1.

**class Test<T extends Number&Runnable>{** //valid

}

**class Test<T extends Number&Runnable&Comparable>**{//valid

}

class Test<T extends Number&String>{//Invalid::At a time mutiple classes

can't be used

}

class Test<T extends Runnable&Number>{//Invalid::First we need to use class

then we need to use interface.

}

**Note::** Defining more than one Type parameter

**class HashMap<K,V>{**

**}**

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

**Generic methods and wild card patterns(?)**

**==================================**

**ArrayList<String> l1=new ArrayList<String>();**

**m1(l1);**

**ArrayList<Integer> l2=new ArrayList<Integer>();**

**m1(l2);**

**ArrayList<Double> l3=new ArrayList<Double>();**

**m1(l3);**

**Use Generic method**

**==================**

1**. public static void m1(ArrayList<String> l)**

|=>It can hold only String type.

eg:: public static void m1(ArrayList<String> l){

l.add("sachin");

l.add(null);

l.add(10.5);//CE: incompatible type

}

Inside the method we can keep only String type and null.

If we keep other type then it would result in compile time error.

2**. public static void m1(ArrayList<?> l){}**

|=> parameter type can be any.

eg:public static void m1(ArrayList<?> l){

l.add("sachin");//invalid

l.add(null);//valid

l.add(10);//invalid

l.add(10.5);//invalid

}

Inside the method we can keep only null.

If we keep other type then it would result in compile time error.

3. **public static void m1(ArrayList<? extends X> l)**

|=> it can be class/interface

if it is class, then it can hold class type or its child type

if it is interface, then it can hold interface type or its implementation class

type.

eg:public static void m1(ArrayList<? extends X> l){

l.add(null);//valid

}

Inside the method we can keep null and other things will be decide by type

of X.

4. **public static void m1(ArrayList<? super X> l)**

|=> it can be class/interface

if it is class, then it can hold class type or its parent type.

if it is interface, then it can hold interface type or its implementation class

parent type.

eg:public static void m1(ArrayList<? super X> l){

l.add(null);//valid

}

Inside the method we can keep null and other things will be decide by type

of X.

eg::

Runnable(I)

| Object(C)

implements |

| |extends

| |

Thread(C)======

Which of the following are valid declarations?

ArrayList<String> al=new ArrayList<String>();//valid

ArrayList<?> al=new ArrayList<String>();//valid

ArrayList<?> al=new ArrayList<Integer>();//valid

ArrayList<?> al=new ArrayList<Double>();//valid

ArrayList<? extends Number> al=new ArrayList<Integer>();//valid

ArrayList<? extends String> al=new ArrayList<Integer>();//CE:incompatible

type

ArrayList<?> al=new ArrayList<? extends String>();//CE: unexpected type

required:class/interface

found : ? extends java.lang.String

ArrayList<?> al =new ArrayList<?>();//CE: unexpected type

found: ?

required: class/interface without bounds.

**Note::** Type parameter can be applied at

a. Class level

b. Method level

**Class level**

**==========**

|=> Type parameter

class Test<T>{

//here T can be of any Type as choosen by the programmer

}

**Method level**

**============**

It is defined to control the argument values of 'T'

**public<T> void m1(T t){}**//valid

**public<T extends Number> void m1(T t){}//**valid

**public<T extends Number&Runnable> void m1(T t){}//**valid

**public <T extends Number&Runnable&Comparable> void m1(T t){}** //valid

public <T extends Number&String> void m1(T t){} //invalid: 2 classes at a time

public <T extends Runnable&Number> void m1(T t){}//invalid: first class and

then interface.

Date: 16/02/2022

Topic: Generics

==========================

**Need of Generics is**

a. To promote type safety.

b. To avoid TypeCasting problems.

**Communication with nongeneric code**

**==================================**

To provide compatible with old version, sun people compromized the concept of generics in very few area's.

eg#1.

import java.util.ArrayList;

public class TestApp {

public static void main(String[] args){

//Configured using jdk1.5 version

ArrayList<String> al=new ArrayList<String>();

al.add("sachin");

//al.add(10);error:no suitable method found add(int)

m1(al);

System.out.println(al);//[sachin, 10, 10.5, true]

al.add(10.5);//error:no suitable method found add(double)

System.out.println(al);//[sachin, 10, 10.5, true,dhoni]

}

//Configured in jdk1.4V

public static void m1(ArrayList al){

al.add(10);

al.add(10.5);

al.add(true);

}

}

Type safety and TypeCasting terminologies in generics are associated with

a. Compiler(correct option)

b. jvm

c. both

d. none of the above

=> Generics concpet is applicable only at the compiler,but not at the JVM.

=> If Generics concept is appliclabe only at the compiler then what is the role of

JVM in generics?

**Ans.** For JVM, there is not concept of generics, it is just one collection object.

Even if we write generics for our Collection class,compiler will use it as

refernce to perform only TypeChecking and TypeCasting.

After the verification is done compiler will remove that generics and it will

be given as just collection class for execution.

**Following syntax are equal for compiler**

**===================================**

1. ArrayList al =new ArrayList<String>();

ArrayList al =new ArrayList<Integer>();

**eg#1.**

**ArrayList al =new ArrayList<String>();**====> ArrayList al =new ArrayList();

**al.add(10);**

**al.add(10.5);**

**al.add(true);**

**System.out.println(al);**

2. ArrayList<String> al =new ArrayList<String>();

ArrayList<String> al =new ArrayList();

**eg#1.**

**ArrayList<String> al =new ArrayList();**===> ArrayList al =new ArrayList();

**al.add("sachin");**

**al.add(10);//error: no suitable method found : add(int)**

**System.out.println(al);**

3. Role played by the compiler during the involvment of Generics in the method

signature

1. Scan the code by referring to generics for typesafety and typecasting

issues.

2. remove the generics after scanning.

3. Once again scan the code without refering to generics

**eg#1.**

**public void m1(ArrayList<Integer> i){}**

**public void m1(ArrayList<String> s){}**

Output

======

CE: error: name clash: m1(ArrayList<String>) and m1(ArrayList<Integer>) have the same erasure

**Generics**

**=======**

1. Objective of Generics

2. TypeParmeter

a. Class Level

b. Method Level

3. Generic method for Paramters

4. Generic Code working in NonGeneric Area.

5. Role played by the Compiler in case of Generics.

**Concurrent Collection**

**=====================**

Normal Collection was introduced in 1.2v

a. Collection

b. List ================> ArrayList,LinkedList,Vector,Stack

c. Set ================> HashSet

a. SortedSet

b. NavigableSet

d. Queue ===============> PriorityQueue,BlockingPriorityQueue

e. Map ================> HashMap,Dictionary,Hashtable,Properties

a. SortedMap

b. NavigableMap

Date: 17/02/2022

Topic: ConcurrentCollection

==========================

**ConcurrentCollection**

What is the need of ConcurrentCollection?

1. Traditional Collection Object like ArrayList,LinkedList,HashSet,HashMap is accessed by Multiple Threads simultaneously and there may be a chance of "DataIncosistencyProblem".

Since they are accessed by multiple threads simultaneously they are not

"ThreadSafe".

b. Already Existed Collection Objects like a Vector,Hashtable

,synchronizedList(),synchronizedSet(),synchronizedMap() are "ThreadSafe",

but performance is very low as the locking mechanism is not good.

c. In case of Normal Collection like ArrayList,.... if one thread is iterating on

Collection Object and if Other thread tries to update the same Collection

Object it would result in and exception called "ConcurrentModification".

Becoz of the above mentioned problem,these collection objects are not suitable for "MultiThreadingEnvironment".To resolve this problem SUNMS introduced a new Set of Collection called "ConcurrentCollection" in JDK1.5.

**Note:**

ConcurrentCollection is part of java.util.concurrent.\*.

**Proof of getting ConcurrentModificationException**

**================================================**

**import java.util.ArrayList;**

**import java.util.Iterator;**

**public class TestApp extends Thread {**

**static ArrayList<String> al=new ArrayList<String>();**

**@Override**

**public void run(){**

**try{**

**Thread.sleep(2000);//2sec**

**}**

**catch (InterruptedException e){}**

**System.out.println("Child thread is updating the list");**

**al.add("nanith");**

**}**

**public static void main(String[] args) throws Exception{**

**al.add("sachin");**

**al.add("kohli");**

**al.add("ABD");**

**TestApp t=new TestApp();**

**t.start();**

**Iterator itr=al.iterator();**

**while(itr.hasNext()){**

**String data =(String)itr.next();**

**System.out.println("Main thread is iterating the list : "+data);**

**Thread.sleep(3000);**

**}**

**System.out.println(al);**

**}**

**}**

**Output**

Main thread is iterating the list : sachin

Child thread is updating the list

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:911)

at java.util.ArrayList$Itr.next(ArrayList.java:861)

at TestApp.main(TestApp.java:28)

**The limitation of Traditional Collection is overcomed in ConcurrentCollections**

1. Concurrent collection is ThreadSafe.

2. Performance of ConurrentCollection is better than NormalCollection becoz

of different locking mechansim.

3. In case of ConcurrentCollection simultaneously read and update operation

can be performed so their is no "ConcurrentModificationException".

**ConcurrentCollection classes**

**============================**

1. ConcurrentHashMap

2. CopyOnWriteArrayList

3. CopyOnWriteArraySet

**ConcurrentMap(I)**

It is a child interface of Map(I).

**It has 3 methods**

**================**

public Object putIfAbsent(Object key,Object value);

//put the entry into the Map,only if key is not present. if key already exists then

it wont replace the value associated with new key.

**eg#1.**

**import java.util.concurrent.ConcurrentHashMap;**

**import java.util.HashMap;**

**public class TestApp {**

**public static void main(String[] args) {**

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

**hm.put(10,"sachin");**

**hm.put(18,"kohli");**

**hm.put(10,"afridi");**

**System.out.println(hm);//{18=kohli,10=afridi}**

**ConcurrentHashMap<Integer,String>chm=new ConcurrentHashMap<Integer,**

**String>();**

**chm.putIfAbsent(10,"sachin");**

**chm.putIfAbsent(18,"kohli");**

**chm.putIfAbsent(10,"afridi");**

**System.out.println(chm);//{18=kohli,10=sachin}**

**}**

**}**

**public boolean remove(Object key, Object value);**

//remove would delete a particular entry only if key and value is matched, if

value is not matched with a key then that particular entry will not be remove.

**eg#1.**

**import java.util.concurrent.ConcurrentHashMap;**

**import java.util.HashMap;**

**public class TestApp {**

**public static void main(String[] args) {**

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

**hm.put(10,"sachin");**

**hm.put(18,"kohli");**

**System.out.println(hm);//{18=kohli,10=sachin}**

**hm.remove(10);**

**System.out.println(hm);//{18=kohli}**

**ConcurrentHashMap<Integer,String>chm=new ConcurrentHashMap**

**<Integer,String>();**

**chm.putIfAbsent(10,"sachin");**

**chm.putIfAbsent(18,"kohli");**

**System.out.println(chm);//{18=kohli,10=sachin}**

**chm.remove(10,"afridi");**

**System.out.println(chm);//{18=kohli,10=sachin}**

**chm.remove(10,"sachin");**

**System.out.println(chm);//{18=kohli}**

**}**

**}**

**public boolean replace(Object key, Object value,Object value);**

//it will replace the value associated with a key only if both key and value is

matched otherwise it wont replace.

**eg#1.**

**import java.util.concurrent.ConcurrentHashMap;**

**import java.util.HashMap;**

**public class TestApp {**

**public static void main(String[] args) {**

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

**hm.put(10,"sachin");**

**hm.put(18,"kohli");**

**System.out.println(hm);//{18=kohli,10=sachin}**

**hm.replace(10,"afridi");**

**System.out.println(hm);//{18=kohli,10=afridi}**

**System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");**

**ConcurrentHashMap<Integer,String>chm=new ConcurrentHashMap<Integer**

**,String>();**

**chm.putIfAbsent(10,"sachin");**

**chm.putIfAbsent(18,"kohli");**

**System.out.println(chm);//{18=kohli,10=sachin}**

**chm.replace(10,"afridi","miller");**

**System.out.println(chm);//{18=kohli,10=sachin}**

**chm.replace(10,"sachin","Messi");**

**System.out.println(chm);//{18=kohli,10=Messi}**

**}**

**}**

**ConcurrentHashMap**

**================**

* **Underlying datastructure** => Hashtable
* **locking mechanism** => not on the entire object, it is on part of

object(bucket level)

* **Read Operation** => Mulitple threads can read no lock required
* **Update Operation** => will be decided by bucket level/concurrency level.
* **DefaultConcurrency level** => 16(can be changed as per user

requirement).

* **null**  => not allowed as both key and value
* **ConcurrentModification** => yes it is possible,no Excpetion.

**eg#1.**

**import java.util.concurrent.ConcurrentHashMap;**

**import java.util.HashMap;**

**import java.util.Set;**

**import java.util.Iterator;**

**public class TestApp extends Thread{**

**//static**

**ConcurrentHashMap<Integer,String>chm=new ConcurrentHashMap<Integer,**

**String>();**

**static HashMap<Integer,String> chm=new HashMap<Integer,String>();**

**@Override**

**public void run(){**

**try{**

**Thread.sleep(2000);//2sec**

**}**

**catch (InterruptedException e){}**

**System.out.println("Child thread is updating the list");**

**chm.put(99,"gayle");**

**}**

**public static void main(String[] args) throws Exception {**

**chm.put(10,"sachin");**

**chm.put(18,"kohli");**

**chm.put(17,"ABD");**

**chm.put(7,"dhoni");**

**TestApp t=new TestApp();**

**t.start();**

**Set s= chm.keySet();**

**Iterator itr=s.iterator();**

**while(itr.hasNext()){**

**Integer data =(Integer)itr.next();**

**System.out.println("Main thread is iterating the list : "+**

**data+"---->"+chm.get(data));**

**Thread.sleep(3000);**

**}**

**System.out.println(chm);**

**}**

**}**

Date: 18/02/2022

**ConcurrentCollection**

**=================**

**Map(I)**

**|**

**|extends**

**|**

**ConcurrentMap(I)**

**|**

**|implements**

**|**

**ConcurrentHashMap(C)** => No ConcurrentModificationException

putIfAbsent(k,v)

remove(k,v)

replace(k,v,v)

**Types of Constructor**

**===================**

**ConcurrentHashMap hmp=ConcurrentHashMap();**

**ConcurrentHashMap hmp=ConcurrentHashMap(int capacity);**

**ConcurrentHashMap hmp=ConcurrentHashMap(int capacity,int fillratio);**

**ConcurrentHashMap hmp=ConcurrentHashMap(int capacity,int fillratio**

**,int concurrencylevel);**

**ConcurrentHashMap hmp=ConcurrentHashMap(Map m);**

**eg#1.**

**import java.util.concurrent.ConcurrentHashMap;**

**import java.util.Set;**

**import java.util.Iterator;**

**public class TestApp{**

**public static void main(String[] args) throws Exception {**

**ConcurrentHashMap<Integer,String>chm=new ConcurrentHashMap<Integer**

**,String>();**

**chm.put(10,"sachin");**

**chm.put(7,"dhoni");**

**Iterator itr= chm.keySet().iterator();**

**chm.put(9,"lara");**

**while(itr.hasNext()){**

**Integer i=(Integer)itr.next();**

**System.out.println(i+"---> "+chm.get(i));**

**Thread.sleep(3000);**

**}**

**System.out.println(chm);**

**}**

**}**

*Reason*

In case of ConcurrentHashMap iterator creates a Read only copy of Map object and iterates over that copy,if any changes to the map happened after getting the iterator won't be reflected in the read only copy object.

In case of normal HashMap the above program would result in "ConcurrentModificationException".

**Difference b/w HashMap and ConcurrentHashMap?**

HashMap => Not Thread safe.

ConcurrentHashMap => Thread safe because of differnt locking

mechanism(bucket level lock)

HashMap => one thread is iterating and at the same time modification

is not possible it would result in

"ConcurrentModificationException".

ConcurrentHashMap => one thread is iterating and at the same time

modification is possibleit won't result in

"ConcurrentModif icationException".

HashMap => Performance is low becoz of more waiting time.

ConcurrentHashMap => Performance is high becoz of less waiting time.

HashMap => Iterator would result in "FailFast".

ConcurrentHashMap => Iterator is of "FailSafe".

HashMap => 1.2v

ConcurrentHashMap => 1.5v

**Difference b/w ConcurrentHashMap,synchronizedMap and Hashtable?**

ConcurrentHashMap => Thread safety by getting lock at the bucket level

SynchronizedMap => Thread safety by appling lock at object level

Hashtable => Thread safety by appling lock at object level

ConcurrentHashMap =>Since lock is at bucket level so no

ConcurrentModificationException.

SynchronizedMap =>Since lock is Object level so

ConcurrentModificationException.

Hashtable =>Since lock is at Object level so

ConcurrentModificationException.

ConcurrentHashMap =>Iterator is "FailSafe".

SynchronizedMap =>Iterator is "FailFast".

Hashtable =>Iterator is "FailFast".

ConcurrentHashMap => Read operation is performed without lock,only for

update operation lock is required only at the bucket

level (concurrency level).

SynchronizedMap => Every read and write operation required lock.

Hashtable => Every read and write operation required lock.

ConcurrentHashMap =>null is not allowed for both key and value.

SynchronizedMap =>null is allowed for both key and value.

Hashtable =>null is not allowed for both key and value.

ConcurrentHashMap =>JDK1.5

SynchronizedMap =>JDK1.2

Hashtable =>JDK1.0

**CopyOnWriteArrayList**

**====================**

Collection(I)

|

List(I)

|

|

CopyOnWriteArrayList(c)

**package ::** java.util.concurrent.CopyOnWriteArrayList(c)

=> It is a Thread Safe Version of ArrayList as the Name indicates

CopyOnWriteArrayList, Creates a Cloned Copy of Underlying ArrayList for

Every Update Operation at Certain Point Both will Synchronized

Automatically Which is taken Care by JVM Internally.

=> As Update Operation will be performed on cloned Copy there is No Effect for

the Threads which performs Read Operation.

=> It is Costly to Use because for every Update Operation a cloned Copy will be

Created.

Hence CopyOnWriteArrayList is the Best Choice if Several Read Operations

and Less Number of Write Operations are required to Perform.

=> Insertion Order is Preserved.

=> Duplicate Objects are allowed.

=> Heterogeneous Objects are allowed.

=> null Insertion is Possible.

=> It implements Serializable, Clonable and RandomAccess Interfaces.

=> While One Thread iterating CopyOnWriteArrayList, the Other Threads are

allowed to Modify and we won’t get ConcurrentModificationException. That

is iterator is FailSafe.

=> Iterator of ArrayList can Perform Remove Operation but Iterator of

CopyOnWriteArrayList can’t Perform Remove Operation. Otherwise we will

get RuntimeException Saying UnsupportedOperationException.

**eg#1.**

**import java.util.concurrent.CopyOnWriteArrayList;**

**import java.util.Iterator;**

**import java.util.ArrayList;**

**public class TestApp{**

**public static void main(String[] args) throws Exception {**

**CopyOnWriteArrayList<Integer>cowl=new CopyOnWriteArrayList<Integer>();**

**cowl.add(5);**

**cowl.add(20);**

**cowl.add(13);**

**cowl.add(15);**

**System.out.println(cowl);[5,20,13,15]**

**Iterator itr=cowl.iterator();**

**while(itr.hasNext()){**

**Integer i=(Integer)itr.next();**

**if(i%2==0)**

**itr.remove();//**RE: UnSupportedOperationException.

**}**

**System.out.println(cowl);**

**}**

**}**

In the above pgm,if i replace CopyOnWriteArrayList with Traditional ArrayList then the ouput would be

[5,20,13,15]

[5,13,15]

**eg#2.**

**import java.util.concurrent.CopyOnWriteArrayList;**

**import java.util.Iterator;**

**import java.util.ArrayList;**

**public class TestApp extends Thread{**

**static CopyOnWriteArrayList<Integer>cowl=new CopyOnWriteArrayList**

**<Integer>();**

**@Override**

**public void run(){**

**try{**

**Thread.sleep(2000);**

**}**

**catch (InterruptedException ie){}**

**System.out.println("Child thread is updating the list");**

**cowl.add(25);**

**}**

**public static void main(String[] args) throws Exception {**

**cowl.add(5);**

**cowl.add(20);**

**cowl.add(13);**

**cowl.add(15);**

**TestApp t= new TestApp();**

**t.start();**

**Iterator itr=cowl.iterator();**

**while(itr.hasNext()){**

**Integer i=(Integer)itr.next();**

**System.out.println("Main Thread iterating on List : "+i);**

**Thread.sleep(3000);**

**}**

**System.out.println(cowl);**

**}**

**}**

**Output**

Main Thread iterating on List : 5

Child thread is updating the list

Main Thread iterating on List : 20

Main Thread iterating on List : 13

Main Thread iterating on List : 15

[5, 20, 13, 15, 25]

If we replace ConcurrentArrayList with ArrayList then the ouput would be

Main Thread iterating on List : 5

Child thread is updating the list

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.AbstractList$Itr.checkForComodification(AbstractList.java:372)

at java.util.AbstractList$Itr.next(AbstractList.java:343)

at TestApp.main(TestApp.java:29)

**Constructors of CopyOnWriteArrayList**

**====================================**

1.CopyOnWriteArrayList cowl=new CopyOnWriteArrayList();

2.CopyOnWriteArrayList cowl=new CopyOnWriteArrayList(Collection c);

3.CopyOnWriteArrayList cowl=new CopyOnWriteArrayList(Object[] o);

**Methods of CopyOnWriteArrayList**

**===============================**

1. public boolean addIfAbsent(Object obj);

//Object will be added into collection depending only if it is not

available

**eg#1.**

**import java.util.concurrent.CopyOnWriteArrayList;**

**import java.util.Iterator;**

**import java.util.ArrayList;**

**public class TestApp extends Thread{**

**public static void main(String[] args) throws Exception {**

**CopyOnWriteArrayList cowl=new CopyOnWriteArrayList();**

**cowl.add("A");**

**cowl.add("B");**

**cowl.add("A");**

**System.out.println(cowl);//[A,B,A]**

**cowl.addIfAbsent("C");**

**cowl.addIfAbsent("A");**

**System.out.println(cowl);//[A,B,A,C]**

**}**

**}**

2. public int addAllAbsent(Collection c);

//Collection will be added into CopyOnWriteArrayList only if the

elements of the Collection are not a part of CopyOnWriteArrayList.

**eg#1.**

**import java.util.concurrent.CopyOnWriteArrayList;**

**import java.util.Iterator;**

**import java.util.ArrayList;**

**public class TestApp extends Thread{**

**public static void main(String[] args) throws Exception {**

**ArrayList<String> l=new ArrayList<String>();**

**l.add("A");**

**l.add("B");**

**System.out.println(l);//[A,B]**

**CopyOnWriteArrayList<String>cowl=new CopyOnWriteArrayList <String>();**

**cowl.add("A");**

**cowl.add("C");**

**System.out.println(cowl);//[A,C]**

**cowl.addAll(l);**

**System.out.println(cowl);//[A,C,A,B]**

**ArrayList<String> l1=new ArrayList<String>();**

**l1.add("A");**

**l1.add("D");**

**cowl.addAllAbsent(l1);**

**System.out.println(cowl);//[A,C,A,B,D]**

**}**

**}**

**Date: 18/02/2022**

**Topic: ConcurrentCollection**

**============================**

**Difference b/w ArrayList and CopyOnWriteArrayList?**

ArrayList => It is not Thread Safe.

CopyOnWriteArrayList => It is Thread Safe.

ArrayList => If one Thread,tries to read and another Thread tries to modify

the Object it would result in "ConcurrentModificationException".

CopyOnWriteArrayList => If one Thread,tries to read and another Thread tries

to modify the Object then no "ConcurrentModificationException".

ArrayList => Iterator is FailFast

CopyOnWriteArrayList => Iterator is FailSafe.

ArrayList => Iterator can be used to perform remove operation.

CopyOnWriteArrayList => If we use remove operation it would result in

"UnsupportedOperation".

ArrayList => JDK1.2V

CopyOnWriteArrayList => JDK1.5v

**Difference b/w CopyOnWriteArrayList,synchronizedList and Vector?**

CopyOnWriteArrayList => It is Thread safe,as every update operation is

performed on seperate cloned copy object

synchronizedList => It is Thread safe, where at a time only one thread is

allowed to operate.

Vector => It is Thread safe, where at a time only one thread is allowed

to operate.

CopyOnWriteArrayList => One thread performing read operation,

simultaneously other thread can perform update

operation it won't result in

"ConcurrentModificationException".

synchronizedList => One thread performing read operation,simultaneously

other thread can't perform update operation it would

result in "ConcurrentModificationException".

Vector => One thread performing read operation,simultaneously other

thread can't perform update operation it would result in "ConcurrentModificationException".

CopyOnWriteArrayList => Iterator is FailSafe.

synchronizedList => Iterator is FailFast.

Vector => Iterator is FailFast.

CopyOnWriteArrayList => Iterator can't perform remove operation it would

result in "UnsupportedOperationException".

synchronizedList => Iterator can perform remove operation.

Vector => Iterator can perform remove operation.

CopyOnWriteArrayList => JDK1.5V

synchronizedList => JDK1.2V

Vector => JDK1.0V

CopyOnWriteArraySet

===================

Collection(1.2V)

|

|

Set(1.2V)

|

|

CopyOnWriteArraySet(1.5v)

1. It is a Thread Safe version of Set.

2. Internally implemented by using CopyOnWriteArrayList.

3. Insertion order is preserved.

4. Null insertion are not allowed.

5. Multiple threads are allowed to perform read operation simultaneouly, where

as if it has to perform update operation ,but seperated copy will be created and

then updation happens.

6. In real time application,if mulitple update operation is required then it is not

suggestible to use CopyOnWriteArraySet as it decreases the performance of

the application.

7. While one Thread is performing read operation,other thread if it tries to peform

update operation then it wont result in "Concurrentmodification" becoz the

modification will happen in seperate copy not on the acutal object.

8. While one Thread is performing read operation and if the other thread tries to

perform remove operation then it would result in

"UnSupportedOperationException".

**Constructor associated with CopyOnWriteArraySet**

**==============================================**

1. **CopyOnWriteArraySet cows =new CopyOnWriteArraySet();**

//creates an empty CopyOnWriteArraySet

2. **CopyOnWriteArraySet cows =new CopyOnWriteArraySet(Collection c);**

//To convert any Collection Object to CopyOnWriteArraySet

**Note::** No special methods is a part of CopyOnWriteArraySet, every method of

Collection and Set interface are only available inside

CopyOnWriteArraySet.

**CopyOnWriteArraySet cows= new CopyOnWriteArraySet();**

**cows.add("A");**

**cows.add("A");**

**cows.add(null);**

**cows.add(10);**

**cows.add("D");**

**cows.add(true);**

**System.out.println(cows);**

**Difference b/w Iteratable and Iterator?**

syntax: for(each item: target){

}

//target should be array/iterable Object.

**Note:** String,StringBuilder,StringBuffer objects are not iterable so we can't use

foreach loop on these Objects.

Iteratable => It is related to foreach loop.

Iterator => It is realated to collection.

Iteratable => To iterate on a Object,that object should be of Iterable type.

Iterator => We use iterator to get every object from Collection type.

Iteratable => It is a part of java.lang package

Iterator => It is a part of java.util package

Iteratable => contains only one method iterator()

Iterator => contains three methods next(),hasNext(),remove()

Iteratable => Introduced in 1.5v

Iterator => Introduced in 1.2v

**Difference b/w CopyOnWriteArraySet and synchronizedSet?**

CopyOnWriteArraySet => Thread safety as every update operation is performed

on cloned copy of Object.

synchronizedSet => Thread safety becoz at a time only one thread is allowed

to operate on a Object.

CopyOnWriteArraySet => If one thread is iterating on a object,if other thread

tries to change the structure of the Collection it won't

result in "ConcurrentModification Exception".

synchronizedSet => If one thread is iterating on a object,if other thread tries to

change the structure of the Collection it would result in

"ConcurrentModification Exception".

CopyOnWriteArraySet => Iterator is FailSafe.

synchronizedSet => Iterator is FaileFast.

CopyOnWriteArraySet => Iterator can perform read only and update operation,if

it tries to perform remove operation it would result in

"UnSupportedOperationException".

synchronizedSet => Iterator can read and remove operation.

CopyOnWriteArraySet => JDK1.5V

synchronizedSet => JDK1.2V

**FailFast vs FailSafe Iterator**

**=============================**

FailFast => This iterator is used to iterate the object from the Collection.

When iteration is happening, internally on the Iterable Object, jvm will

Maintains some flag called "MOD" and then it peforms the iteration.

If the flag value is changed JVM will internally generate an Exception

called "ConcurrentModificationException".

**eg#1.**

**import java.util.concurrent.CopyOnWriteArraySet;**

**import java.util.Iterator;**

**import java.util.ArrayList;**

**public class TestApp extends Thread{**

**public static void main(String[] args){**

**ArrayList<String> al = new ArrayList<String>();**

**al.add("sachin");**

**al.add("saurav");**

**al.add("dhoni");**

**al.add("rahul");**

**al.add("kohli");**

**System.out.println(al);**

**Iterator itr=al.iterator();**

**while (itr.hasNext()){**

**String data=(String)itr.next();**

**System.out.println(data);**

**al.add("afridi");//**RE: ConcurrentModificationException

|=> because of change in flag value.

**}**

**System.out.println(al);**

**}**

**}**

**FailSafe**

**========**

Using this iterator, we can iterate the Collection.

During iteration if we perform any change over the collection Object,those

changes won't be reflected into old memeory,rather the changes gets reflected in

the "ClonedObject" due to which the flag value won't be changed and it won't

result in "ConcurrentModificationException".

**import java.util.concurrent.CopyOnWriteArrayList;**

**import java.util.Iterator;**

**import java.util.ArrayList;**

**public class TestApp extends Thread{**

**public static void main(String[] args){**

**CopyOnWriteArrayList<String>cowl=new CopyOnWriteArrayList<String>();**

**cowl.add("sachin");**

**cowl.add("saurav");**

**cowl.add("dhoni");**

**cowl.add("rahul");**

**cowl.add("kohli");**

**System.out.println(cowl);**

**Iterator itr=cowl.iterator();**

**while (itr.hasNext()){**

**String data=(String)itr.next();**

**System.out.println(data);**

**cowl.add("afridi");**

**}**

**System.out.println(cowl);**

**}**

**}**

**output**

[sachin, saurav, dhoni, rahul, kohli]

sachin

saurav

dhoni

rahul

kohli

[sachin, saurav, dhoni, rahul, kohli, afridi, afridi, afridi, afridi, afridi]

**Difference b/w FailFast and FailSafe?**

**Property FailFast FailSafe**

Does it throw Yes No

ConcurrentModificationExcetpion

Cloned copy will be created No Yes

Memory problems No Yes

Example ArrayList,Vector,... CopyOnWriteArrayList,

ConcurrentHashMap,...

**Topics to be discussed**

**======================**

1. Wrapper classes(Boxing and AutoBoxing)

2. Access modifiers

3. package creation and working with jars

4. JVM architecure

5. Garbage Collector

6. Inner classes

7. enum

=====================

1. IO

2. Networking

========================

JDK1.8 features

JDK1.9 features(modules replacement for jars)

JDK10,11,12,13,14.

**Date: 22/02/2022**

**Topic: WrapperClass**

**==================**

**Topics to be discussed**

**======================**

1. **Wrapper classes(Boxing and AutoBoxing)**

All wrapper classes are a part of java.lang package.

**What is the need of Wrapper class?**

To wrap primtive type of data also as a Object,so that we can handle primitives

also as "Object".

It defines few utility methods which are required for primitives.

**Constructors**

**============**

Almost all the wrapper classes have 2 constructors which can take 2 arguments

a. String argument constructor

b. Primitive type constructor

**eg#1. Integer i=new Integer("10");**

**Integer i=new Integer(10);**

**Integer i=new Integer("ten");//**RE:NumberFormatException

**There are 8 wrapper classes**

1. Byte 2.Short 3.Integer 4. Long 5.Float 6.Double for all these classes Number

is the Parent.

7. Character 8.Boolean for these classes Object class is the Parent.

**WrapperClass Constructor**

------------- ---------------

Byte String,byte

Short String,short

Integer String,int

Long String,long

Float String,float,double

Double String,double

=====================================================

Character char

Boolean boolean,String

In case of Boolean Wrapper class, if we pass String as the input,by default the data would be treated as true only for non-case sensitive data of "true",for all the other cases it is treated as false.

eg: "true" => true

"True" => true

"tRuE" => true

"TRUE" => true

eg#1.

Boolean b1=new Boolean(true);//valid: boolean type: true

Boolean b2=new Boolean(false);//valid: boolean type:false

Boolean b3=new Boolean(True);//CE

Boolean b4=new Boolean(False);//CE

MCQ

===

eg#1.

Boolean b5=new Boolean("yes");

System.out.println(b5);//false

Boolean b6=new Boolean("no");

System.out.println(b6);//false

System.out.println(b5.equals(b6));//true

Option

A. CE

B. RE

C. true(correct answer)

D. false

E. None of the above

eg#2

Boolean b7=new Boolean("Left");

System.out.println(b7);//false

Boolean b8=new Boolean("Right");

System.out.println(b8);//false

System.out.println(b7.equals(b8));//true

Note::

* equals() in all Wrapper classes is Overriden from the Object class to check the content of the data.
* toString() in all Wrapper classes is Overriden from the Object class to print the data present inside the Object.
* All Wrapper classes are also Immutable.
* In String class also equals() and toString() is Overriden in the same manner.
* Wrapper class and String class is also Immutable.
* Wrapper class whose immediate parent is Object are Character,Boolean
* Wrapper class whose immediate parent is not Object are

:Byte,Short,Integer,Long,Float,Double.

**Utility Methods**

**===============**

These are static methods through which the programmer can call them directly using classname.

**1.valueOf()**

**2.xxxxValue()**

**3.parseXXXX()**

**4.toString()**

**1.valueOf()**

This method is used to convert String/primitive type to Wrapper Object.

Except Character class, in every wrapper class this valueOf() method is

available.

**public static WrapperType valueOf(String s)**

**eg#1.**

**Integer i=Integer.valueOf("10");**

**System.out.println(i);**

**System.out.println(i.toString());**

**Double d=Double.valueOf("10.5");**

**Boolean b=Boolean.valueOf("sachin");**

**System.out.println(b);//false**

2.

**public static WrapperType valueOf(String s,int radix)**

|=> 2 (0,1)

|=> 8 (0,...7)

|=> 10(0,...9)

|=> 16(0,....9,A...F)

|=> 36(0,....9,A...Z)

Except Character class, in every wrapper class this valueOf(String,int) method is available.

eg#1.

Integer i=Integer.valueOf("1111",37);

System.out.println(i);//RE:NumberFormatException : 37 > Character.MAX\_RADIX

3. **public static Wrappertype valueOf(primitivetype data)**

In every class including wrapper class this method is available.

eg#1.

Character c=Character.valueOf('c');

Integer i=Integer.valueOf(10);

Double d=Double.valueOf(10.5);

Boolean b=Boolean.valueOf(true);

Date: 22/02/2022

Topic: WrapperClass

============================

**Topics to be discussed**

**======================**

1. **Wrapper classes(Boxing and AutoBoxing)**

All wrapper classes are a part of java.lang package.

**xxxValue()** => Totally 38 methods

This method is used to convert the wrapper Object to primitive type.

All Number wrapper classes have the following methods

1. byteValue() 2. shortValue() 3. intValue() 4. longValue()

5. floatValue() 6. doubleValue()

W.r.t to Character and Boolean wrapper class we have the following method

1. char charValue() => Character class.

2. boolean booleanValue() => Boolean class.

**eg#1.**

**public class TestApp {**

**public static void main(String[] args){**

**Integer i=Integer.valueOf("130");**

**System.out.println(i.byteValue());**//result= minvalue +(result-

maxvalue-1)

-128 + (130-127-1) = -126

**System.out.println(i.shortValue());//130**

**System.out.println(i.intValue());//130**

**System.out.println(i.longValue());//130**

**System.out.println(i.floatValue());//130.0**

**System.out.println(i.doubleValue());//130.0**

**}**

**}**

**eg#2.**

**public class TestApp {**

**public static void main(String[] args){**

**Character c=Character.valueOf('a');**

**System.out.println(c.charValue());//a**

**Boolean b=Boolean.valueOf("nitin");**

**System.out.println(b.booleanValue());//false**

**}**

**}**

String class => Immutable and he is final.

Wrapper class => Immutable and he is final.

**Just like String class,Wrapper class can we make userdefined class also Immutable?**

**Ans.** Yes we can make userdefined class also Immutable.

**final public class TestApp {**

**int i;**

**public TestApp(int i){**

**this.i=i;**

**}**

**public TestApp modify(int i){**

**if (this.i == i){**

**return this;**

**}else{**

**return new TestApp(i);**

**}**

**}**

**public static void main(String[] args){**

**TestApp t1=new TestApp(10);**

**TestApp t2=t1.modify(10);**

**System.out.println(t1);**

**System.out.println(t2);**

**System.out.println(t1==t2);//true**

**System.out.println();**

**TestApp t3= t1.modify(100);**

**System.out.println(t1);**

**System.out.println(t3);**

**System.out.println(t1==t3);//false**

**}**

**}**

**Output**

TestApp@7852e922

TestApp@7852e922

true

TestApp@7852e922

TestApp@4e25154f

false

**parseXXX()**

**==========**

=> This method is used to convert the String Object into primitive type.

=> Every wrapper class, excluding the Character class contains parseXXX() to

convert String to Primitive type.

**form-1**

public static primitivetype parseXXX(String s)throws NumberFormatException

**form-2**

public static primitivetype parseXXX(String s,int radix) throws NumberFormatException

|=> 2 to 36

**eg#1.**

**public class TestApp {**

**public static void main(String[] args){**

**int i= Integer.parseInt("10");**

**System.out.println(i);**

**double d=Double.parseDouble("25.5");**

**System.out.println(d);**

**boolean b= Boolean.parseBoolean("false");**

**System.out.println(b);**

**int i2= Integer.parseInt("1111",2);**

**System.out.println(i);//15**

**}**

**}**

toString()

=> This method is used to convert Wrapper class or primitive type to String.

1. **public String toString()**

//It is overriden to give implementation such that it always prints the

data present in the Object.

**eg#1.**

**public class TestApp {**

**public static void main(String[] args){**

**Integer i=Integer.valueOf("10");**

**System.out.println(i);**

**System.out.println(i.toString());**

**}**

**}**

2. **public String toString(primitivetype data);**

This method is a part of all Wrapper classes, including character class.

**eg#1.**

**public class TestApp {**

**public static void main(String[] args){**

**String s1= Integer.toString(10);**

**String s2= Double.toString(10.5);**

**String s3= Boolean.toString(true);**

**String s4= Character.toString('a');**

**System.out.println(s1);**

**System.out.println(s2);**

**System.out.println(s3);**

**System.out.println(s4);**

**}**

**}**

**Output**

10

10.5

true

a

3. **public String toString(primtivietype data,int radix);**

|=> 2 to 36

These method are a part of Integer and Long class only.

**eg#1.**

**public class TestApp {**

**public static void main(String[] args){**

**String s1=Integer.toString(1111,2);**

**System.out.println(s1);**

**String s2=Integer.toString(1111,8);**

**System.out.println(s2);**

**String s3=Integer.toString(1111,16);**

**System.out.println(s3);**

**}**

**}**

**Output**

10001010111

2127

457

**form-4**

**======**

**public static String toHexString(int);**

**public static String toOctalString(int);**

**public static String toBinaryString(int);**

**eg#1.**

**public class TestApp {**

**public static void main(String[] args){**

**String s1=Integer.toString(1111,2);**

**System.out.println(s1);**

**String s2=Integer.toString(1111,8);**

**System.out.println(s2);**

**String s3=Integer.toString(1111,16);**

**System.out.println(s3);**

**System.out.println();**

**String s4=Integer.toBinaryString(1111);**

**System.out.println(s4);**

**String s5=Integer.toOctalString(1111);**

**System.out.println(s5);**

**String s6=Integer.toHexString(1111);**

**System.out.println(s6);**

**}**

**}**

**Output**

10001010111

2127

457

10001010111

2127

457