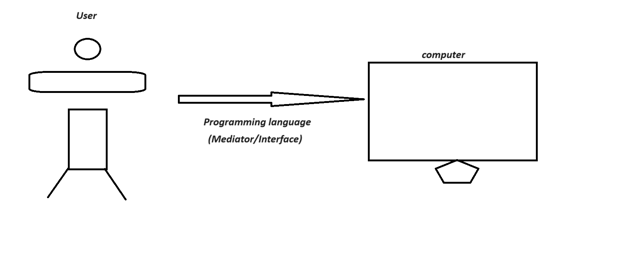
Programming language

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

Diagram: introduction1.1



A language which is used to communicate between user and computer is called programming language.

Programming language acts like a mediator or interface between user and computer.

**Java**

**====**

* **Object oriented programming language.**
* **Platform independent programming language.**
* **Case sensitive programming language.**
* **Strongly typed checking language.**
* **High level programming language.**
* **Open source programming language.**
* **1995** -->**James Gosling** --> Sun Micro System (Oracle Corporation)
* **Java software** -->**JDK** software

**C**

**==**

* **Procedure oriented programming language.**
* **Platform dependent programming language.**
* **Case sensitive programming language.**
* **Loosely typed checking language.**
* **Middle level language (LOW + HIGH).**

**Interview Questions**

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

Q) What is Java?

**Java is**a**object oriented, platform independent, case sensitive, strongly typed checking, high level , open source programming language**developed by James Gosling in the year of 1995.

Q) What is the **difference between Python and Java?**

**Python Java**

**--------- --------**

It is developed by Guido Van Rossum. It is developed by James Gosling.

It is a product of Microsoft. It is a product of Oracle Corporation.

It is a scripting language. It is a object oriented programming language.

It is a interpreted language. It is a compiled language.

It contains PVM. It contains JVM.

It is a dynamically typed language. It is a statically typed language.

ex: ex:

i = 10; int i = 10;

Performance is low. performance is high.

There is less security. It is highly secured.

**Note:**

**-----**

.py file --------------------------------> computer

(high level language)

.java file ----------------> .class file -----------> computer

(High level language) (**Byte Code**)

**Note:**

**----**

PVM

.pyfile ----------------------------------------> computer/machine

compiler JVM

.java file ----------> .class file --------------> JRE

(byte Code)

**Project**

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

Technically, **a project is a collection of modules.**

We have following list of modules.

ex:

* + registration module
  + admin module
  + login module
  + report generation
  + payment module

and etc.

Every project contains two domains.

**1) Technical domain**

**---------------------**

Technical domain **describes using which technology we developed our project.**

ex:

Java

**2) Functional domain**

**--------------------**

Functional domain **describes state of a project.**

ex:

Healthcare domain

Banking domain

ERP domain

Insurance domain

and etc.

Interview Questions

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

Q) How many **modules are there in Java?**

We have three modules in Java.

**1) JSE/J2SE (Java Standard Edition)**

**2) JEE/J2EE (Java Enterprise Edition)**

**3) JME/J2ME (Java Micro Edition**

**Comments in Java**

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

Comments are **created for documentation purpose.**

Comments are **used to improve readability of our code.**

It is highly recommended to use comments in our regular programming.

Comments will not display in output because they won't compiled by the compiler.

In java, we have two types of comments.

1) Single Line Comment

2) Multiple Line Comment

**1) Single Line Comment**

**-----------------------**

It is used to comment a single line.

ex:

**// comment here**

**2) Multiple Line Comment**

**-----------------------**

It is used to comment multiple lines.

ex:

**/\***

**-**

**- comment here**

**-**

**\*/**

**ex:**

**---**

**//class declaration**

class Test

{

**//main method**

public static void main(String[] args)

{

**//variable declaration**

int x = 10;

**//output stmt**

System.out.println(x);

}

}

**Interview Questions**

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

Q) What is the **difference between JDK, JRE and JVM ?**

**JDK**

**----**

JDK stands for Java Development Kit.

JDK is a **installable software** which consist of Java Runtime Environment( **JRE**), Java Virtual Machine (**JVM**) , **compiler** (**javac**), **interpreter (java)**,

an **archiever (.jar)** ,**document generator (javadoc)** and **other tools needed for java application development.**

**JRE**

**----**

JRE stands for Java Runtime Environment.

It **provides very good environment to run java applications only**.

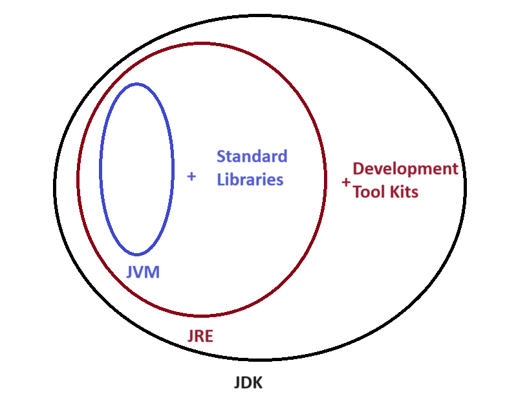
**JVM**

**---**

JVM stands for Java Virtual Machine.

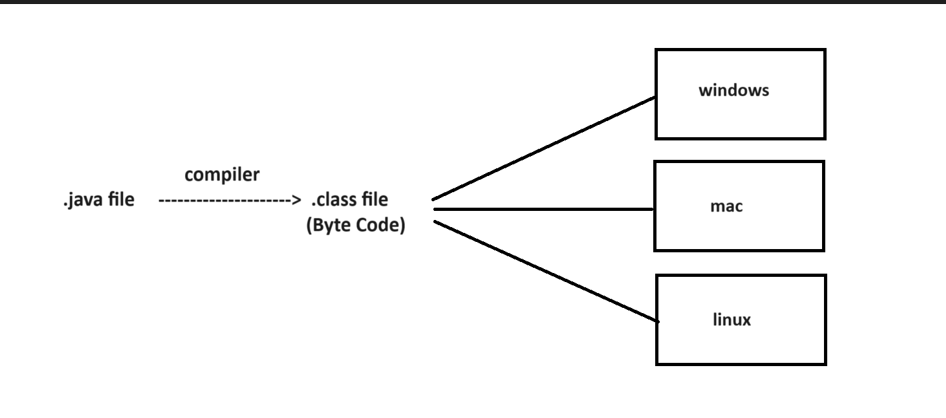
**JVM** is a**interpreter**which is **used to execute our program line by line procedure and** it **converts byte code to machine code.**

**Diagram: class1.2**



Q) Is java platform independent or dependent?

**Java is platform independent.**



Q) Is JVM platform independent or dependent?

It is platform dependent.

Q) A **.class file contains** \_\_\_ code ?

**byte code**

**Naming conventions in Java**

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

In java, **uppercase letter**s will consider as different **and lowercase letters** will **consider as different**.**Hence** we consider **java is a case sensitive programming language.**

As java is a case sensitive, we must and should follow naming conventions for following things.

ex:

classes

interfaces

variables

methods

keywords

packages and

constants

**classes**

**--------**

In java, a class name must and should starts with**uppercase** letter and if it contains multiple words then each inner word must starts with**initcap**.

**ex:**

**predefined classes userdefined classes**

**-------------------- ---------------------**

System Test

FileWriter DemoApp

BufferedReader ExampleApp

File EmployeeDetails

StringBuilder StudentInfo

and etc. and etc.

**interfaces**

**-----------**

In java, an interface name must and should starts with **uppercase** letter and if it contains

multiple words then each inner word must and should starts with **initcap.**

**ex:**

**predefined interfaces userdefined interfaces**

**--------------------- ---------------------**

Runnable ITest

Serializable IDemoApp

ListIterator IStudentInfo

Enumeration IDepartment

and etc. and etc.

**variables**

**-----------**

In java, an variable name **must and should starts with lowercase** letter and if it contains

multiple words then **each inner word** must and should starts with **initcap** ex:

**predefined variables userdefined variables**

**--------------------- ---------------------**

out i

in empId

err studName

length deptNo

and etc. and etc.

**Methods**

**--------**

In java, **a method name** must and should **starts with lowercase** letter and if it contains multiple words then each **inner word** must and should **starts with initcap.**

ex:

**predefined methods userdefined methods**

**------------------- --------------------**

getPriority() getStudentInfo()

setName() setBillDetails()

getClass() calculateBillAmt()

hashCode() and etc.

toString()

and etc.

**keywords**

**--------**

In java, **all keywords we need to declare under lowercase letters only.**

ex:

**predefined keywords**

**-------------------**

if , else, do, while, break, public, static, void and etc.

**packages**

**----------**

In java, **all packages we need to declare under lowercase letters only.**

ex:

**predefined packages userdefined packages**

**----------------- --------------------**

java.lang com.ihub.www

java.util com.google.www

java.time com.qt.www

java.text and etc.

java.util.stream

java.sql

and etc.

**constants**

**-----------**

In java, all constants we need to declare under uppercase letters only.

ex:

**predefined constants userdefined constants**

**----------------- -------------------**

MAX\_PRIORITY LIMIT=10;

MIN\_PRIORITY

NORM\_PRIORITY

MAX\_VALUE

MIN\_VALUE

and etc.

**Assignment**

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

**1) class :** QualityThought

**2) Interface :** IQualityThought

**3) variable :** qualityThought

**4) Method :** qualityThought()

**5) package :** com.qualitythought.www

**6) constant :** QUALITY\_THOUGHT

**Interview Questions**

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

Q) Which package is a **default package in java?**

**java.lang** package

Q) What is **package?**

**A package is a collection of classes and interfaces**.

**History of Java**

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

* In 1990, Sun Micro System took one project to develop a software called consumer electronic device which can be controlled by a remote like setup box.
* That time project was called Stealthproject and later it is renamed to Green project.
* James Gosling, Mike Sheradin and Patrick Naughton were there to develop the project. They metin a place called Aspan/Colarado to start the work with Graphic System. James Gosling thought to use C and C++ languages to develop the project.
* But the problem what they have faced is
* C and C++ languages are system dependent. Then James Gosling decided , why don't we createour own programming language which is system independent.
* In 1991, They have developed a programming language called an OAK. OAK means strength, itselfis a coffee seed name and it is a national tree for many contries like Germany , France,
* USA and etc.
* Later in 1995, they have renamed OAK to Java. Java is a island of an Indonasia where first coffee of seed was produced. During the development of project they were consuming lot of coffee's.Hence the symbol of java is a cup of coffee with saucer.

**Interview Questions**

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

Q) Who is the **creator of java?**

**James Gosling**

Q) **Java originally known as** \_\_\_?

**OAK**

Q) **In which year java was developed?**

**In 1995**

Q) What are the **features of Java?**

We have following important features in java.

**1) Simple**

**2) Object oriented**

**3) Platform independent**

**4) Architecture neutral**

**5) Multithreaded**

**6) Robust**

**7) Portable**

**8) High secured**

**9) Dynamic**

**10)Distributed**

**and etc.**

Java

=====

Version : Java 11

Software : JDK 11

Creator : James Gosling

Vendor : Oracle Corporation

Website : www.oracle.com/in/java

Open source : Open source

Tutorial :www.javatpoint.com

www.w3school.com

www.tutorialspoint.com

www.javaus.com

www.dzone.com

and etc.

Download link :

https://drive.google.com/file/d/1GtRLHXK4y3s97BH2UcYiJPNBaROR1DBV/view?usp=drive\_link

Steps to setup environmental variables

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

step1:

-------

Make sure JDK 11 installed successfully.

step2:

-----

Copy a "lib" directory from java\_home folder.

ex:

C:\Program Files\Java\jdk-11\lib

step3:

----

Paste "lib" directory in environmental variables.

ex:

right click to This PC --> properties --> Advanced system settings -->

environmental variables --> user variables --> click to new button -->

variable name : CLASSPATH

variable value : C:\Program Files\Java\jdk-11\lib;

---> ok.

system variables --> click to new button -->

variable name : path

variable value : C:\Program Files\Java\jdk-11\bin; --> ok --> ok --> ok.

step4:

-----

Open the command prompt and check environmental variables done perfectly or not.

ex:

cmd>javap

cmd> java -version

Steps to develop first java application

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

step1:

-----

Make sure JDK 11 installed successfully.

step2:

-----

Make sure environmental setup done perfectly.

step3:

------

Open the notepad and develop Hello World program.

ex:

class Test

{

public static void main(String[] args)

{

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

}

}

step4:

-----

Create a "javaprog" folder in 'E' drive.

step5:

-----

Save above program with same name as class inside "javaprog" location.

step6:

------

Open the command prompt from "javaprog" location.

step7:

------

Compile above program by using below command.

ex:

javac Test.java

|

filename

step8:

------

Run above program by using below command.

ex:

java Test

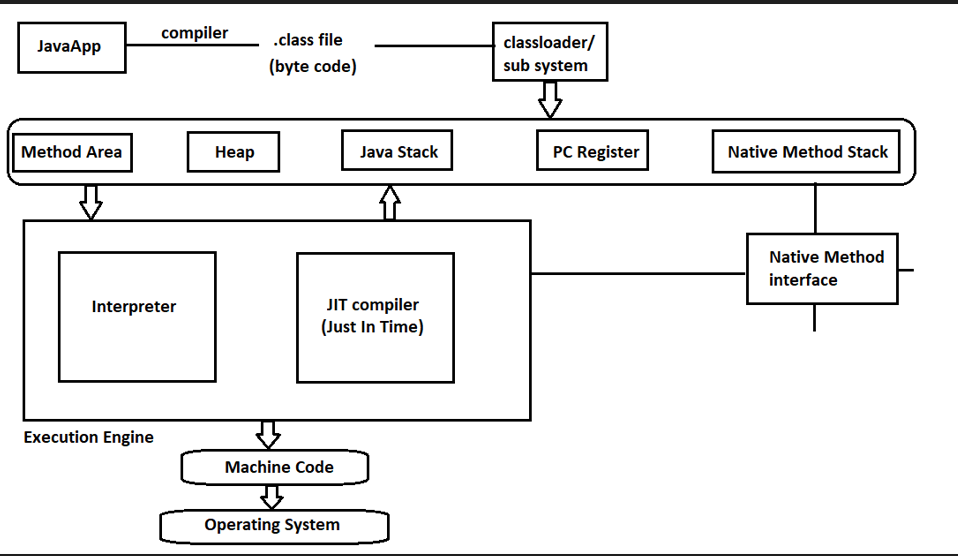
|

classname

Internal Architecture of JVM

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

Diagram: class5.1



* Java program contains java code instructions.Once if we compile java code instructions converts to byte code instructions in .class file.
* Now JVM will invoke one module called classloader or sub system to load all the byte code instructions from .class file.
* The work of classloader is to check these byte code instructions are proper or not.
* If they are not proper then it will refuse the execution. If they are proper then it will allocates the memories.

We have five types of memories.

1) Method Area

------------

It contains code of a class, code of a variable and code of a method.

2) Heap

--------

Our object creations will store in heap area.

3) Java Stack

--------------

Java methods will store in method area.

To execute those methods we required some memory and that memory will be allocated in java stack.

4) PC Register

--------------

It is a program counter register which is used to track the address of an instructions.

5) Native Method Stack

-----------------------

Java methods will execute in method area.

Similarly native methods will execute in native method stack.

But to execute native methods we required a program called Native method interface.

Execution Engine

----------------

Execution engine contains interpreter and JIT compiler.

Whenever JVM loads byte code instructions from .class file , it will uses interpreter and JIT compiler.

Interpreter is used to execute our program line by line procedure.

JIT compiler is used to increase the execution speed of our program.

Finally , JVM converts byte code to machine code.

Interview Question

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

Q) Explain about internal Architecture of java/jvm?

A ---above it has explained----

Q) What is native method in java?

A method which is developed by using some other language is called native method.

Q) How many memories are there in java?

We have five memories in java.

1) Method Area

2) Heap

3) Java Stack

4) PC Register

5) Native Method Stack

Q) What is JIT compiler?

It is a part of a JVM which is used to increase the execution speed of our program.

Q) How many classloaders are there in java?

We have three classloaders in java.

1) Bootstrap classloader

2) Extension classloader

3) Application/System classloader

Identifiers

===========

A name in java is called identifier.

It can be variable name, method name, class name or label name.

ex:

class Test

{

public static void main(String[] args)

{

int x = 10;

System.out.println(x);

}

}

Here Test, main ,String , args, x , System are identifiers.

Rules to declare an identifiers

--------------------------------

Rule1:

------

Identifier will accept following characters.

ex:

A-Z

a-z

0-9

\_

$

Rule2:

------

If we take other characters then we will get compile time error.

ex:

int emp$al;

String stud\_Name;

double emp#Fee; //invalid

Rule3:

------

Identifier must and should starts with alphabet, underscore or dollar symbol but not

with digits.

ex:

int \_empId; //valid

int $alary;//valid

int a1234; //valid

int 1abcd; //invalid

Rule4:

------

Every identifier is a case sensitive.

ex:

int number;

int NUMBER;

int NuMbEr;

Rule5:

-----

We can't take reserved words as an identifier name.

ex:

int if; //invalid

int else; //invalid

Rule6:

-----

There is no length limit for an identifier but it is not recommanded to take more then

15 characters.

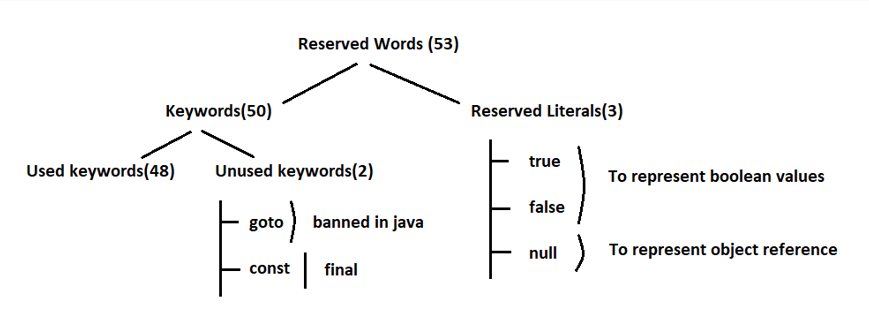
Reserved Words

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

There are some identifiers which are reserved to associate some funcationality or meaning such type of identifiers are called reserved words.

Java supports 53 reserved words and it is divided into two types.

Diagram: class6.1i



Used keywords with respect to class

-------------------------------------

import

package

interface

enum

class

extends

implements

Used keywords with respect to object

----------------------------------

new

instanceof

this

super

Used keywords with respect to datatypes

-------------------------------

byte

short

int

long

float

double

boolean

char

Used keywords with respect to return type

------------------------

void

Used keywords with respect to modifiers

------------------------------

default

public

private

protected

static

final

abstract

synchronized

strictfp

transient

volatile

Used keywords with respect to flow control

--------------------------

if

else

switch

case

break

continue

do

while

for

Used keywords with respect to exception handling

--------------------

try

catch

throw

throws

finally

assert

Datatypes

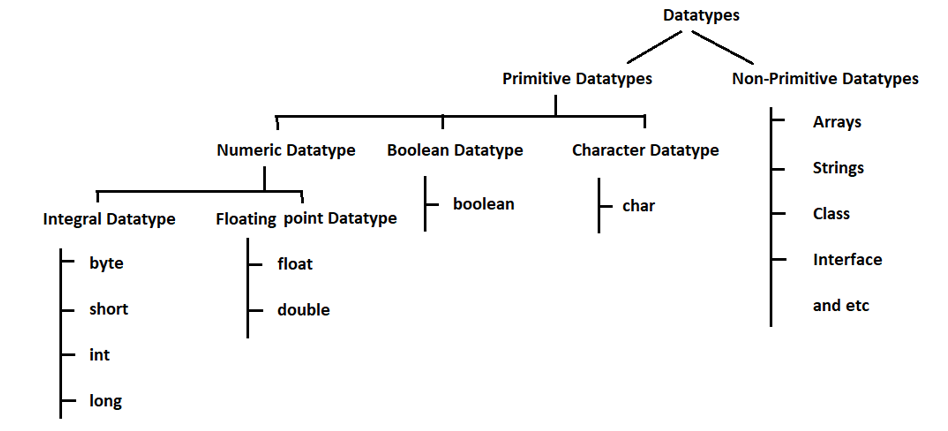
==========

Datatype describes what type of value we want to store inside a variable.

Datatype also tells how much memory has to be created for a variable.

In java, we have two types of datatypes

**Diagram: class6.2**



byte

-------

It is a smallest datatype in java.

Size : 1 byte (8 bits)

Range: -128 to 127 (-2^7 to 2^7-1)

ex:

1) byte b=10;

System.out.println(b); // 10

2) byte b=10.5;

System.out.println(b); // C.T.E

3) byte b="hi";

System.out.println(b); // C.T.E

short

------

It is a rarely used datatype in java.

Size : 2 bytes (16 bits)

Range : -32768 to 32767 (-2^15 to 2^15-1)

ex:

---

1) byte b=10;

short s=b;

System.out.println(s); // 10

2) short s=true;

System.out.println(s); // C.T.E

3) short s=10.56;

System.out.println(s); // C.T.E

int

-----

It is a mostly used datatype in java.

Size : 4 bytes (32 bits)

Range : -2147483648 to 2147483647 (-2^31 to 2^31-1)

ex:

1) int i=10.56;

System.out.println(i); // C.T.E

2) int i=true;

System.out.println(i); // C.T.E

3) int i="hi";

System.out.println(i); // C.T.E

4) int i='a';

System.out.println(i); // 97

Note:

-----

In java, every character has a universal unicode value.

ex:

a --> 97

A --> 65

long

--------

If int datatype is not enough to hold large value then we need to use long datatype.

Size : 8 bytes (64 bits)

Range : -2^63 to 2^63-1

ex:

1) long l=10.56;

System.out.println(l); // C.T.E

2) long l="hi";

System.out.println(l); // C.T.E

3) long l=true;

System.out.println(l); // C.T.E

4) long l='a';

System.out.println(l); // 97

**float double**

------------- ----------------

If we need 4 to 6 decimal point of accuracy then If we need 14 to 16 decimal point of accuracy

we need to use float. then we need to use double.

Size: 4 bytes (32 bits) Size: 8 bytes (64 bits)

Range : -3.4e38 to 3.4e38 Range: -1.7e308 to 1.7e308

To declare a float value we need to suffix with To declare a double value we need to suffix

'f' or 'F'. with 'd' or 'D'.

ex: ex:

10.56f 10.56d

ex:

---

1) float f=10.56f;

System.out.println(f); // 10.56

2) float f=10;

System.out.println(f); // 10.0

3) float f='A';

System.out.println(f); // 65.0

4) float f="true";

System.out.println(f); // C.T.E

5) float f=false;

System.out.println(f); // C.T.E

ex:

---

1) double d=10.56d;

System.out.println(d); // 10.56

2) double d=10;

System.out.println(d); // 10.0

3) double d='A';

System.out.println(d); // 65.0

4) double d="true";

System.out.println(d); // C.T.E

5) double d=false;

System.out.println(d); // C.T.E

**boolean**

**---------**

It is used to represent boolean values either true or false.

Size : (Not Applicable) (1-bit)

Range: (Not Applicable)

ex:

1) boolean b="true";

System.out.println(b); // C.T.E

2) boolean b=TRUE;

System.out.println(b); // C.T.E

3) boolean b=true;

System.out.println(b); // true

**char**

**-------**

It is a single character which is enclosed in a single quotation.

Size : 2 bytes (16 bits)

Range: 0 to 65535

ex:

1) char ch='a';

System.out.println(ch); //a

2) char ch="p";

System.out.println(ch); // C.T.E

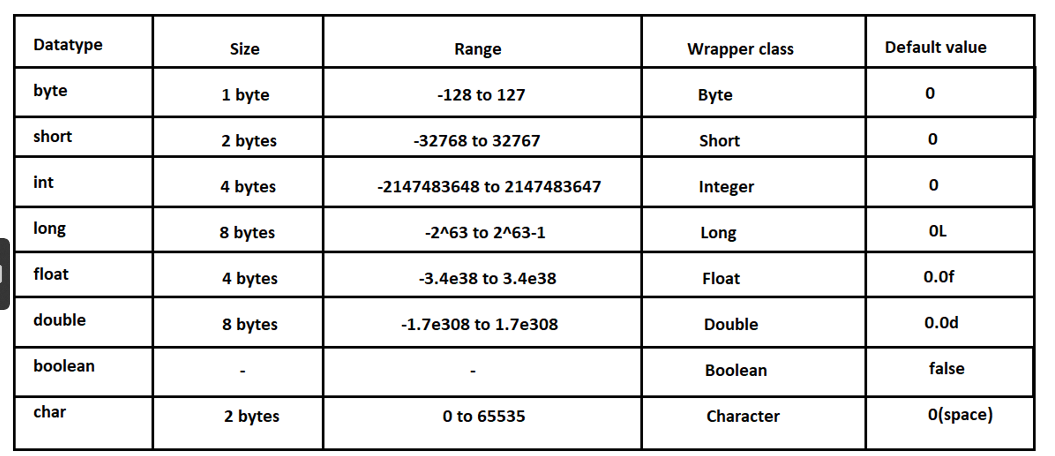
3) char ch='ab';

System.out.println(ch); // C.T.E

4) char ch=65;

System.out.println(ch); // A

**Diagram: class7.1**



Interview Question

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

Q) Is java purely object oriented or not?

No, java will not consider as purely object oriented programming language because it does not

support many OOPS concepts like multiple inheritance, operator overloading and more ever we

depends upon primitive datatypes which are non-objects.

Q) Write a java program to display range of byte datatype?

byte : -128 to 127

class Test

{

public static void main(String[] args)

{

System.out.println(Byte.MIN\_VALUE);

System.out.println(Byte.MAX\_VALUE);

}

}

Q) Write a java program to display range of short datatype?

short : -32768 to 32767

ex:

class Test

{

public static void main(String[] args)

{

System.out.println(Short.MIN\_VALUE);

System.out.println(Short.MAX\_VALUE);

}

}

Types of variables

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

A name which is given to a memory location is called variable.

Purpose of variable is used to store the data.

In java, we have two types of variables.

1) Primitive variables

2) Reference variables

1) Primitive variables

-----------------------

Primitive variables are used to represent primitive values.

2) Reference variables

----------------------

Reference variables are used to represent object.

ex:

Student s=new Student();

|

reference variable

Based on the position and execution these variables are divided into three types.

1) Instance variables / Non-static variables

2) Static variables / Global variables

3) Local variables / Temperory variables / Automatic variables

1) Instance variables

----------------------

A value of a variable which is varied(changes) from object to object is called instance variable.

Instance variable will be created at the time of object creation and it will destroy at the time of

object destruction.Hence scope of instance variable is same as scope of an object.

Instance variable will store in heap area as a part of an object.

Instance variable must and should declare immediately after the class but not inside methods,

blocks and constructors.

Instance variable we can access directly from instance area but we can't access directly from

static area.

To access instance variable from static area we need to create object reference.

ex:1

-----

class Test

{

//instance variable

int i=10;

public static void main(String[] args)

{

System.out.println(i); // C.T.E

}

}

ex:2

----

class Test

{

//instance variable

int i=10;

public static void main(String[] args)

{

Test t=new Test();

System.out.println(t.i); // 10

}

}

Note:

----

If we won't initialize any value to instance variable then JVM will initialized default values.

ex:3

-----

class Test

{

//instance variable

boolean b;

public static void main(String[] args)

{

Test t=new Test();

System.out.println(t.b);// false

}

}

ex:4

----

class Test

{

//instance variable

int i=10;

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test();

System.out.println(t1.i);//10

System.out.println(t2.i);//10

t1.i=100;

System.out.println(t1.i);//100

System.out.println(t2.i);//10

}

}

ex:5

-----

class Test

{

public static void main(String[] args)

{

//calling

Test t=new Test();

t.m1();

}

//non-static method

public void m1()

{

System.out.println("instance-method");

}

}

If we write the logic in a seperate instance method then we can reuse the bussiness logic.

ex:6

----

class Test

{

public static void main(String[] args)

{

Test t=new Test();

t.m1();

t.m1();

t.m1();

}

//non-static method

public void m1()

{

System.out.println("instance-method");

}

}

2) static variables

--------------------

A value of a variable which is not varied(changes) from object to object is called static variable.

A static variable will be created at the time of classloading and it will destroy at the time of

classunloading.Hence scope of static variable is same as scope of a .class file.

Static variables will store in method area.

Static variable must and should declare immediately after the class using static keyword but not inside methods, blocks and constructors.

Static variable we can access directly from static area as well as instance area.

Static variable we can access by using object reference and class name.

ex:1

-----

class Test

{

//static variable

static int i=10;

public static void main(String[] args)

{

System.out.println(i); //10

Test t=new Test();

System.out.println(t.i); //10

System.out.println(Test.i);//10

}

}

Note:

-----

If we won't initialize any value to static variable JVM will initialized default values.

ex:2

-----

class Test

{

//static variable

static String s;

public static void main(String[] args)

{

System.out.println(s); // null

Test t=new Test();

System.out.println(t.s); // null

System.out.println(Test.s); // null

}

}

ex:3

-----

class Test

{

//static variable

static int i=10;

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test();

System.out.println(t1.i);//10

System.out.println(t2.i);//10

t1.i=100;

System.out.println(t1.i); //100

System.out.println(t2.i); //100

}

}

ex:4

-----

class Test

{

public static void main(String[] args)

{

m1();

Test t=new Test();

t.m1();

Test.m1();

}

//static method

public static void m1()

{

System.out.println("static-method");

}

}

3) Local variables

--------------------

To meet temperory requirements a programmer will declare some variables inside methods, blocks

and constructors such type of variables are called local variables.

Local variable will be created at the time of execution block and it will destroy when execution

block is executed.Hence scope of local variable is same as scope of a execution block where it is

declared.

Local variables will store in Java stack.

ex:1

-----

class Test

{

public static void main(String[] args)

{

//local variable

int i=10;

System.out.println(i); //10

}

}

Note:

-------

If we won't initialize any value to local variable then JVM will not initialized default values.

ex:2

----

class Test

{

public static void main(String[] args)

{

//local variable

int i;

System.out.println(i); //

}

}

o/p:

C.T.E : variable i might not have been initialized

Interview Question

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

Q) Jack and John both are best friends in a town.While going to school they saw one begger.Jack and john both decided to help that needy person.Jack gave Rs.50 from his pocket money and John gave Rs.100 from his bag.Write a java console to find out total contribution for a poor guy?

class Test

{

public static void main(String[] args)

{

sum();

}

//static method

public static void sum()

{

int a=50;

int b=100;

int c=a+b;

System.out.println(c);

}

}

**Main method**

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

Our program contains main method or not.

Either it is properly declared or not.

It is not a responsibility of a compiler to check. It is a liability of a JVM to look for main method always at runtime.

If JVM won't find main method then it will throw one runtime error called main method not found.

JVM always look for main method with following signature.

signature:

--------

public static void main(String[] args)

If we perform any changes in above signature then JVM Will throw one runtime error called main method not found.

Q) Explain main method in java?

**public**

**------**

JVM wants to call main method from anywhere.

**static**

**------**

JVM wants to call main method without using object reference.

**void**

**-----**

Main method does not return anything to JVM.

**main**

**----**

It is a identifier given to a main method.

**String[] args**

**-----------**

It is a command line argument.

We can perform following changes in main method.

1) Order of modifiers is not important. Incase of public static we can declare static public also.

ex:

static public void main(String[] args)

2) We can change String[] in following acceptable formats.

ex:

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

3) We can change String[] with var-arg parameter.

ex:

public static void main(String... args)

4) We can replace args with any java valid identifier.

5) Main method will accept following modifiers.

ex:

synchronized

strictfp

final

**Command line arguments**

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

Arguments which are passing through command prompt such type of arguments are called command line

arguments.

In command line arguments we need to pass input values at runtime command.

ex:

javac Test.java

java Test 101 raja M 1000.0

| | | |\_\_\_\_ args[3]

| | |\_\_\_\_\_\_\_\_ args[2]

| |\_\_\_\_\_\_\_\_\_\_\_\_\_ args[1]

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ args[0]

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println(args[0]);

System.out.println(args[1]);

System.out.println(args[2]);

System.out.println(args[3]);

}

}

System.out.println()

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

It is a output statement in java.

Whenever we want to display any data or user defined statements then we need to use output stmt.

syntax:

-----

static variable

|

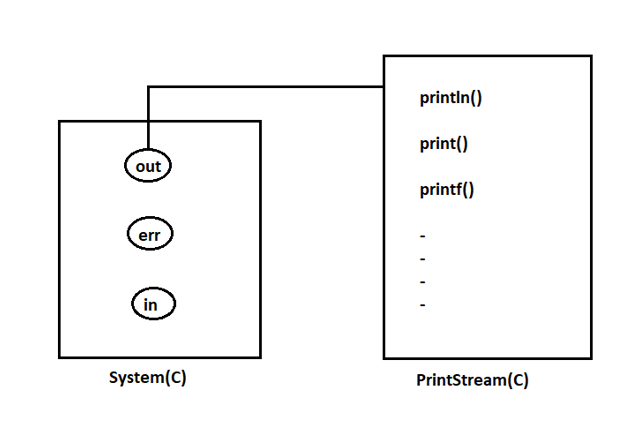
System.out.println();

| |

predefined predefined method

final class

Diagram: class10.1



ex:

----

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

System.out.print("stmt2");

System.out.printf("stmt3");

}

}

Fully Qualified Name

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

Fully qualified name means we need to declare a class or interface along with package name.

It is used to increase the readability of our code.

ex:

---

class Test

{

public static void main(String[] args)

{

java.util.Date d =new java.util.Date();

System.out.println(d);

}

}

Import statements

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

Whenever we use import statement we should not use fully qualified name.

Using shortname also we can achieve.

In java, we have three import statements.

1) Explicit class import

2) Implicit class import

3) Static import

1) Explicit class import

-------------------------

This type of import statement is highly recommanded to use because it will improve readability of our code.

ex:

---

import java.time.LocalDate;

import java.time.LocalTime;

class Test

{

public static void main(String[] args)

{

LocalDate date=LocalDate.now();

System.out.println(date);

LocalTime time=LocalTime.now();

System.out.println(time);

}

}

2) Implicit class import

--------------------

This type of import statement is not recommanded to use because it will reduce the readability of our code.

ex:

---

import java.time.\*;

class Test

{

public static void main(String[] args)

{

LocalDate date=LocalDate.now();

System.out.println(date);

LocalTime time=LocalTime.now();

System.out.println(time);

}

}

static import

----------------

Using static import we can call static members (static variables and static methods) directly.

Often use of static import makes our program complex and unreadable.

ex:

---

import static java.lang.System.\*;

class Test

{

public static void main(String[] args)

{

out.println("stmt1");

out.println("stmt2");

out.println("stmt3");

}

}

ex:

---

import static java.lang.System.\*;

class Test

{

public static void main(String[] args)

{

out.println("stmt1");

exit(0);

out.println("stmt2");

}

}

Typecasting in Java

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

The process of converting from one datatype to another datatype is called typecasting.

In java, typecasting can be done in two ways.

1) Implicit typecasting

2) Explicit typecasting

1) Implicit typecasting

-----------------------

If we want to store small value into a bigger variable then we need to use implicit typecasting.

A compiler is responsible to perform implicit typecasting.

There is no possibility to loss the information.

It is also known as Widening or Upcasting.

We can perform implicit typecasting as follow.

ex:

byte --> short

-->

int --> long --> float --> double

-->

char

ex:

---

class Test

{

public static void main(String[] args)

{

byte b=10;

int i=b;

System.out.println(i);//10

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

char ch='a';

long l=ch;

System.out.println(l);//97

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

double d=i;

System.out.println(d);//10.0

}

}

1) Explicit typecasting

-----------------------

If we want to store big value into a small variable then we need to use explicit typecasting.

A programmer is responsible to perform implicit typecasting.

There is a possibility to loss the information.

It is also known as Norrowing or Downcasting.

We can perform explicity typecasting as follow.

ex:

byte <-- short

<--

int <-- long <-- float <-- double

<--

char

ex:

byte <-- short

<--

int <-- long <-- float <-- double

<--

char

ex:

---

class Test

{

public static void main(String[] args)

{

double d=10.5d;

int i=(int)d;

System.out.println(i); // 10

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=65;

char ch=(char)i;

System.out.println(ch); // A

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=130;

byte b=(byte)i;

System.out.println(b); // -126

}

}

Types of blocks in java

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

A block is a set of statements which is enclosed in a curly bracesi.e {}.

In java, we have three types of blocks.

1) Instance block

2) Static block

3) Local block

1) Instance block

----------------

Instance block is used to initialize the values to instance variables.

Instance block must and should declare immediately after the class but not inside methods and constructors.

Instance block will execute when create an object.

We can declare instance block as follow.

syntax:

//instance block

{

-

- //set of statements

-

}

ex:

----

class Test

{

//instance block

{

System.out.println("instance-block");

}

public static void main(String[] args)

{

System.out.println("main-method");

}

}

o/p:

main-method

ex:

---

class Test

{

//instance block

{

System.out.println("instance-block");

}

public static void main(String[] args)

{

System.out.println("main-method");

Test t=new Test();

}

}

o/p:

main-method

instance-block

ex:

---

class Test

{

//instance block

{

System.out.println("instance-block");

}

public static void main(String[] args)

{

Test t1=new Test();

System.out.println("main-method");

Test t2=new Test();

}

}

o/p:

instance-block

main-method

instance-block

ex:

---

class Test

{

//instance variable

int i;

//instance block

{

i=100;

}

public static void main(String[] args)

{

Test t=new Test();

System.out.println(t.i);//100

}

}

2) static block

----------------

Static block is used to initialize the values to static variables.

Static block must and should declare immediately after the class using static keyword but not inside methods and constructors.

Static block will execute at the time of classloading.

We can declare static block as follow.

syntax:

//static block

static

{

-

- //set of statements

-

}

ex:

---

class Test

{

//static block

static

{

System.out.println("static-block");

}

public static void main(String[] args)

{

System.out.println("main-method");

}

}

o/p:

static-block

main-method

ex:

----

class Test

{

//instance block

{

System.out.println("instance-block");

}

//static block

static

{

System.out.println("static-block");

}

public static void main(String[] args)

{

Test t=new Test();

System.out.println("main-method");

}

}

o/p:

static-block

instance-block

main-method

ex:

---

class Test

{

//static variable

static int i;

//static block

static

{

i=200;

}

public static void main(String[] args)

{

System.out.println(i); //200

}

}

3) local block

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

Local block is used to initialize the local variables.

Local block must and should declare inside the methods.

Local block will execute just like a normal statement.

syntax:

------

//local block

{

-

- //set of statements

-

}

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

//local block

{

System.out.println("stmt2");

}

System.out.println("stmt3");

}

}

o/p:

stmt1

stmt2

stmt3

ex:

----

class Test

{

public static void main(String[] args)

{

//local variable

int i;

//local block

{

i=300;

}

System.out.println(i); // 300

}

}

Q) Can we execute java program without main method?

Yes, Till 1.6 version it is possible to execute java program without main method using static block.But from 1.7 version onwards it is not possible to execute java program without main method.

ex:

---

class Test

{

//static block

static

{

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

System.exit(0);

}

}

Operators

==========

Operator is a symbol which is used to perform some operations on operands.

ex:

c = a + b;

Here + and = are operators

Here a, b and c are operands.

It can be arithmetic operation, logical operation, bitwise operation , relational operation and etc.

We have following list of operators in java.

1) Assignment operators

2) Ternary operators

3) Logical operators

4) Bitwise operators

5) Arithmetic operators

6) Relational operators

7) Shift operators

8) Unary operators

1) Assignment operators

-----------------------

class Test

{

public static void main(String[] args)

{

int i=10;

i=20;

i=30;

System.out.println(i);//30

}

}

Note:

-----

Reinitialization is possible in java.

ex:

---

class Test

{

public static void main(String[] args)

{

final int i=10;

i=20;

i=30;

System.out.println(i);//C.T.E

}

}

Note:

-----

Final variables can't be modified.

ex:

----

class Test

{

public static void main(String[] args)

{

int i=1,2,3,4,5;

System.out.println(i); //C.T.E

}

}

ex:

---

class Test

{

//global variable

static int i=10;

public static void main(String[] args)

{

//local variable

int i=20;

System.out.println(i); // 20

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

i+=20; // i = i + 20;

System.out.println(i); // 30

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

i-=20; // i = i - 20;

System.out.println(i); // -10

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

i\*=20; // i = i \* 20;

System.out.println(i); // 200

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=5;

i/=2; // i = i / 2;

System.out.println(i); //2

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=5;

i/=20;

System.out.println(i); //0

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=5;

i%=2;

System.out.println(i); //1

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=5;

i%=20;

System.out.println(i); // 5

}

}

2) Ternary operator

--------------------

syntax:

------

(condition)?value1:value2;

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b=(5>2)?true:false;

System.out.println(b);//true

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=(5>20)?1:0;

System.out.println(i);//0

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

String s=(true)?"Hi":"Bye";

System.out.println(s); // Hi

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

char ch=(false)?'t':'f';

System.out.println(ch); // f

}

}

Q) Write a java program to find out greatest of two numbers?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

//logic

int max=(a>b)?a:b;

System.out.println("Greatest of two numbers is ="+max);

}

}

Q) Write a java program to find out greatest of three numbers?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

System.out.println("Enter the third number :");

int c=sc.nextInt();

//logic

int max=(a>b)?((a>c)?a:c):((b>c)?b:c);

System.out.println("Greatest of three numbers is ="+max);

}

}

3) Logical operators

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

Logical AND operator (&&)

--------------------------

Local AND operator deals with boolean value either true or false.

Truth table

-----------

T T = T

T F = F

F T = F

F F = F

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println(true && true); //true

System.out.println(true && false); //false

System.out.println(false && true); // false

System.out.println(false && false); // false

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b= (5>2) && (6<10);

System.out.println(b);//true

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b= (5>2) && (6<1);

System.out.println(b);//false

}

}

Logical OR operator (||)

-------------------------

Logical OR operator deals with boolean values either true or false.

Truth table

----------

T T = T

T F = T

F T = T

F F = F

ex:

class Test

{

public static void main(String[] args)

{

System.out.println(true || true);//true

System.out.println(true || false);//true

System.out.println(false || true);//true

System.out.println(false || false);//false

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b=(6>10) || (5<2);

System.out.println(b); // false

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b=(6>1) || (5<2);

System.out.println(b); // true

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b=(7>2) && (5<10) || (6>10);

System.out.println(b); // true

}

}

Logical NOT operator (!)

-----------------------

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b= !(5>2);

System.out.println(b); // false

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

boolean b= !(5>20);

System.out.println(b); // true

}

}

How to convert decimal to binary number

----------------------------------------

10 - decimal number

1010 - binary number

2|10

---- 0

2|5

---- 1

2|2

---- 0 ^

1 |

-----------------

1010

How to convert binary to decimal number

--------------------------------------

0101 - binary number

5 - decimal number

0101

<--

1\*1 + 0\*2 + 1\*4 + 0\*8

1 + 0 + 4 + 0 = 5

4) Bitwise operators

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

Bitwise AND operator (&)

--------------------------

Bitwise AND operator deals with binary numbers.

Truth table

-------------

T T = T

T F = F

F T = F

F F = F

ex:

---

class Test

{

public static void main(String[] args)

{

int a=10,b=5;

int c = a & b;

System.out.println(c); //0

}

}

/\*

10 - 1010

5 - 0101

-----------

& - 0000

\*/

ex:

---

class Test

{

public static void main(String[] args)

{

int a=10,b=15;

int c = a & b;

System.out.println(c); //10

}

}

/\*

10 - 1010

15 - 1111

---------

& - 1010 <---

0\*1 + 1\*2 + 0\*4 + 1\*8

0 + 2 + 0 + 8 = 10

\*/

Bitwise OR operator (|)

--------------------------

Bitwise OR operator deals with binary numbers.

Truth table

-------------

T T = T

T F = T

F T = T

F F = F

ex:

---

class Test

{

public static void main(String[] args)

{

int a=2,b=3;

int c = a | b;

System.out.println(c); // 3

}

}

/\*

2 - 0010

3 - 0011

---------

| - 0011 <--

1\*1 + 1\*2 + 0\*4 + 0\*8

1 + 2 + 0 + 0 = 3

\*/

ex:

---

class Test

{

public static void main(String[] args)

{

int a=10,b=5;

int c = a | b;

System.out.println(c); // 15

}

}

/\*

10 - 1010

5 - 0101

----------

| - 1111

\*/

Bitwise XOR operator (^)

--------------------------

Bitwise XOR operator deals with binary numbers.

Truth table

-------------

T T = F

T F = T

F T = T

F F = F

ex:

---

class Test

{

public static void main(String[] args)

{

int a=10,b=15;

int c = a ^ b;

System.out.println(c); // 5

}

}

/\*

10 - 1010

15 - 1111

----------

^ - 0101

\*/

Bitwise NOT operator (~)

-------------------------

ex:

---

class Test

{

public static void main(String[] args)

{

int i=~10;

System.out.println(i); // -11

}

}

ex:

----

class Test

{

public static void main(String[] args)

{

int i=~23;

System.out.println(i); // -24

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=~(-12);

System.out.println(i); // 11

}

}

5) Arithmetic operators

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

% - modules

/ - division

\* - multiplication

+ - addition

- - subtraction

ex:

---

class Test

{

public static void main(String[] args)

{

int i= 6+7/2+6%3+7\*4+9/10+7%20+6-10;

System.out.println(i); //40

}

}

/\*

6 + 7/2 + 6%3 + 7\*4 + 9/10 + 7%20 + 6 - 10

6 + 3 + 0 + 28 + 0 + 7 + 6 - 10

50 - 10

40

\*/

6) Relational operators

------------------------

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println(10 > 20); //false

System.out.println(10 >= 20); //false

System.out.println(10 < 20); //true

System.out.println(10 <= 10); //true

System.out.println(10 == 10);//true

System.out.println(10 == 20);//false

System.out.println(10 != 20); //true

System.out.println(10 != 10); //false

}

}

7) Shift operators

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

Right shift operator (>>)

--------------------------

10 >> 1 = 10/2

10 >> 2 = 10/4

10 >> 3 = 10/8

10 >> 4 = 10/16

ex:

---

class Test

{

public static void main(String[] args)

{

int i= 10 >> 3; // 10/8

System.out.println(i); //1

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i= 100 >>6; //100/ 64

System.out.println(i); //1

}

}

Left shift operator (<<)

--------------------------

10 << 1 = 10\*2

10 << 2 = 10\*4

10 << 3 = 10\*8

10 << 4 = 10\*16

ex:

---

class Test

{

public static void main(String[] args)

{

int i= 10 <<3; // 10\*8 =80

System.out.println(i); // 80

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i= 100 <<2; // 100 \* 2\*2

System.out.println(i); // 400

}

}

8)Unary operators

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

Increment/Decrement operators (++/--)

-----------------------------------

We have two types of increment operators

1) Post increment

ex:

i++;

2) Pre increment

ex:

++i;

We have two types of decrement operators.

1) Post decrement

ex:

i--;

2) Pre decrement

ex:

--i;

POST Increment/Decrement

------------------------

Rule1: First Take

Rule2: Then Change

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

i++;

System.out.println(i); // 11

}

}

ex:

----

class Test

{

public static void main(String[] args)

{

int i=10;

System.out.println(i++); //10

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

int j= i++ + i++; //10 + 11

System.out.println(i+" "+j); //12 21

}

}

ex:

----

class Test

{

public static void main(String[] args)

{

int i=10;

int j=i-- + i--; //10 + 9

System.out.println(i+" "+j); // 8 19

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

int j=i++ + i-- + i++; //10 + 11 + 10

System.out.println(i+" "+j); // 11 31

}

}

Pre increment/decrement

------------------------

Rule1 : First Change

Rule2 : Then Take

ex:

----

class Test

{

public static void main(String[] args)

{

int i=10;

++i;

System.out.println(i); //11

}

}

ex:

----

class Test

{

public static void main(String[] args)

{

int i=10;

System.out.println(++i); // 11

}

}

ex:

----

class Test

{

public static void main(String[] args)

{

int i=10;

int j=++i + ++i; //11 + 12

System.out.println(i+" "+j); //12 23

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

int j=--i + --i; // 9 + 8

System.out.println(i+" "+j); // 8 17

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

System.out.println(i++ + ++i); //10 + 12 = 22

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=100;

100++;

System.out.println(i);//C.T.E

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

System.out.println(++(i++));//C.T.E

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

byte b=127;

b++;

System.out.println(b);//-128

}

}

Control Statements

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

Control statement enables the programmer to control flow of a program.

Control statement allows the programmer to make decisions, to jump from one section of code to another section and to execute the code repeatedly.

In java, we have four control statements.

1) Decision Making statement

2) Selection statement

3) Iteration statement

4) Jump statement

1) Decision Making statement

----------------------------

Decision making statement is used to declare conditions in our program.

Decision making statement is possible by using following ways.

i) if stmt

ii) if else stmt

iii) if else if ladder

iv) nested if stmt

i) if stmt

-----------

It is used to execute the source code only if our condition is true.

syntax:

-------

if(condition)

{

-

- //code to be execute

-

}

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(5>2)

{

System.out.println("stmt2");

}

System.out.println("stmt3");

}

}

o/p:

stmt1

stmt2

stmt3

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(!(5>2))

{

System.out.println("stmt2");

}

System.out.println("stmt3");

}

}

o/p:

stmt1

stmt3

ex:

---

class Test

{

public static void main(String[] args)

{

if(!(5>2))

System.out.println("stmt1");

System.out.println("stmt2");

System.out.println("stmt3");

}

}

o/p:

stmt2

stmt3

ii) if else stmt

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

It will execute the source code either our condition is true or false.

syntax:

------

if(condition)

{

- //code to be execute if cond is true

}

else

{

- //code to be execute if cond is false

}

ex:

----

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(5>10)

{

System.out.println("stmt2");

}

else

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

}

o/p:

stmt1

stmt3

stmt4

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(5>1)

{

System.out.println("stmt2");

}

else

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

}

o/p:

stmt1

stmt2

stmt4

iii) if else if ladder

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

It will execute the source code based on multiple conditions.

syntax:

------

if(cond1)

{

- //code to be execute if cond1 is true

}

else if(cond2)

{

- //code to be execute if cond2 is true

}

else if(cond3)

{

- //code to be execute if cond3 is true

}

else

{

- //code to be execute if all conditions are false

}

ex:

---

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the option :");

int option=sc.nextInt();

if(option==100)

System.out.println("It is a police number");

else if(option==103)

System.out.println("It is enquiry number");

else if(option==108)

System.out.println("It is emergency number");

else

System.out.println("Invalid option");

}

}

iv) nested if stmt

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

If stmt contains another if stmt is called nested if stmt.

syntax:

------

if(condition)

{

if(Condition)

{

-

- //code to be execute

-

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(5>2)

{

System.out.println("stmt2");

if(true)

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

System.out.println("stmt5");

}

}

o/p:

stmt1

stmt2

stmt3

stmt4

stmt5

ex:

-----

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(5>2)

{

System.out.println("stmt2");

if(false)

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

System.out.println("stmt5");

}

}

o/p:

stmt1

stmt2

stmt4

stmt5

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(5>20)

{

System.out.println("stmt2");

if(true)

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

System.out.println("stmt5");

}

}

Q) Write a java program to check given number is positive or negative using nested if stmt?

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

if(n!=0)

{

if(n>0)

{

System.out.println("It is positive number ");

System.exit(0);

}

System.out.println("It is negative number");

}

}

}

Orchasp interview Question

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

Write a java program to calculate costs based on user input. The program

should prompt users to enter the total weight of items(in kilograms) and the shipping destination (domestic or international). for demostic orders, the program should charge Rs.500 for weights upto 5 kg and Rs.100 per additional kg. for international orders, it should charge Rs.1000 for weights upto 5 kg , Rs.200 per additional kg , and a Rs.500 surcharge for weights exceeding 10 kg. print calculated shipping cost.

input:

total weight of items : 11

Shipping Destination : domestic

output:

1100

Explaination

------------

weight <=5 = (Rs.500)

weiht<=11 = (Rs.500 + 600)

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter total weight of item :");

int weight=sc.nextInt(); // 11

System.out.println("Enter shipping destination :");

String destination=sc.next(); // domestic

if(destination.equals("domestic"))

{

if(weight<=5)

{

System.out.println(500);

}

if(weight>5)

{

System.out.println(500 + (weight-5)\*100);

}

}

else if(destination.equals("international"))

{

if(weight<=5)

{

System.out.println(1000);

}

else if(weight>5 && weight<=10)

{

System.out.println(1000 + (weight-5) \* 200);

}

else if(weight>10)

{

System.out.println(1000 + (weight-5) \* 200 + 500);

}

}

}

}

2) Selection statement

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

switch case

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

It will execute the source code based on multiple conditions.

It is similar to if else if ladder.

syntax:

-------

switch(condition)

{

case value1: //code to be execute

break stmt;

case value2: //code to be execute

break stmt;

-

-

default: //code to be execute if all cases are false.

}

ex:

----

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the option :");

int option=sc.nextInt();

switch(option)

{

case 100: System.out.println("It is a police number");

break;

case 103: System.out.println("It is a enquiry number");

break;

case 108: System.out.println("It is a emergency number");

break;

default :System.out.println("Invalid option");

}

}

}

Declaration of break stmt in switch case is optional.If we won't declare break statement then from where our condition is satisfied from there all cases will be executed that state is called fall through state of switch case.

ex:

---

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the option :");

int option=sc.nextInt();

switch(option)

{

case 100: System.out.println("It is a police number");

//break;

case 103: System.out.println("It is a enquiry number");

//break;

case 108: System.out.println("It is a emergency number");

//break;

default :System.out.println("Invalid option");

}

}

}

3) Iteration statement

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

Iteration statement is used to execute the code repeatedly.

Iteration statement is possible by using loops.

We have four types of loops.

i) do while loop

ii) while loop

iii) for loop

iv) for each loop

i) do while loop

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

It will execute the source code how long our condition is true.

syntax:

------

do

{

-

- //code to be execute

-

}while(condition);

ex:

---

class Test

{

public static void main(String[] args)

{

int i=1;

do

{

System.out.print(i+" "); // infinite 1

}

while (i<=10);

}

}

In do while loop, our code will execute atleast for one time either our condition is true or false.

ex:

class Test

{

public static void main(String[] args)

{

int i=11;

do

{

System.out.print(i+" "); // 11

}

while (i<=10);

}

}

3) Iteration statement

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

Iteration statement is used to execute the code repeatedly.

Iteration statement is possible by using loops.

We have four types of loops.

i) do while loop

ii) while loop

iii) for loop

iv) for each loop

i) do while loop

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

It will execute the source code how long our condition is true.

syntax:

------

do

{

-

- //code to be execute

-

}while(condition);

ex:

---

class Test

{

public static void main(String[] args)

{

int i=1;

do

{

System.out.print(i+" "); // infinite 1

}

while (i<=10);

}

}

In do while loop, our code will execute atleast for one time either our condition is true or false.

ex:

class Test

{

public static void main(String[] args)

{

int i=11;

do

{

System.out.print(i+" "); // 11

}

while (i<=10);

}

}

ii) while loop

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

It will execute the source code how long our condition is true.

syntax:

-------

while(condition)

{

-

- //code to be execute

-

}

ex:

----

class Test

{

public static void main(String[] args)

{

int i=1;

while(i<=10)

{

System.out.print(i+" "); // infinite 1

}

}

}

ex:

----

class Test

{

public static void main(String[] args)

{

int i=11;

while(i<=10)

{

System.out.print(i+" "); //nothing

}

}

}

iii) for loop

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

It will execute the source code how long our condition is true.

syntax:

-------

for(initialization;condition;incrementation/decrementation)

{

-

- //code to be execute

-

}

If number of iterations are known by the user then we need to use for loop.

If number of iterations are not known by the user then we need to use while loop.

If number of iterations are not known by the user but code must execute alteast for one time then we need to use do while loop.

ex:

---

class Test

{

public static void main(String[] args)

{

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

{

System.out.print(i+" "); //1 2 3 4 5 6 7 8 9 10

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

for(;;)

{

System.out.print("Hello "); // infinite Hello

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

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

{

System.out.print("Hello ");

i--;

}

}

}

4) Jump Statement

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

Jump statement is used to jump from one section of code to another section.

We have two types of jump statements.

i) break statement

ii) continue statement

i) break statement

-------------------

It is used to break the execution of loops and switch case.

For conditional statement we can use if condition.

ex:

----

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

break;

System.out.println("stmt2");

}

}

o/p:

C.T.E : break outside switch or loop

ex:

----

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(true)

{

break;

}

System.out.println("stmt2");

}

}

o/p:

C.T.E : break outside switch or loop

ex:

---

class Test

{

public static void main(String[] args)

{

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

{

if(i==5)

{

break;

}

System.out.print(i+" "); //1 2 3 4

}

}

}

ii) continue statement

-------------------

It is used to continue the execution of loops.

For conditional statements we can use if condition.

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

continue;

System.out.println("stmt2");

}

}

o/p:

C.T.E : continue outside of loop

ex:

----

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(true)

{

continue;

}

System.out.println("stmt2");

}

}

o/p:

C.T.E : continue outside of loop

ex:

---

class Test

{

public static void main(String[] args)

{

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

{

if(i==5)

{

continue;

}

System.out.print(i+" "); //1 2 3 4 6 7 8 9 10

}

}

}

Various ways to declare the methods in java

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

1) No return type with No argument method

2) No return type with Argument method

3) With return type with No argument method

4) With return type with Argument method

1) No return type with No argument method

------------------------------------------

If we don't have arguments then we need to ask input values inside callie method.

Q) Write a java program to perform sum of two numbers with no returntype with no argument method?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

sum();

sum();

}

//callie method

public static void sum()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

//logic

int c=a+b;

System.out.println("sum of two numbers is ="+c);

}

}

Q) Write a java program to display sum of 10 natural numbers using no returntype with no argument method?

class Test

{

public static void main(String[] args)

{

//caller method

sum();

}

//callie method

public static void sum()

{

int sum=0;

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

{

sum+=i;

}

System.out.println("sum of 10 natural numbers is ="+sum);

}

}

2) No return type with Argument method

-------------------------------------

If we have arguments then we need to ask input values inside main method.

Here number of arguments depends upon number of inputs.

Q) Write a java program to perform sum of two numbers using no returntype with argument method?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();//10

System.out.println("Enter the second number :");

int b=sc.nextInt();//20

//caller method

sum(a,b);

}

//callie method

public static void sum(int a,int b)

{

int c=a+b;

System.out.println("sum of two numbers is ="+c);

}

}

Q) Write a java program to perform cube of a given number using no returntype with argument method?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//caller method

cube(n);

}

//callie method

public static void cube(int n)

{

//int result=(int)Math.pow(n,3);

int result = n\*n\*n;

System.out.println("cube of a given number is ="+result);

}

}

Assignment

==========

Q) Write a java program to check given number is prime or not?

3) With returntype with No argument method

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

Returntype is completely depends upon output datatype.

Q) Write a java program to perform sum of two numbers using with returntype with no argument method?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

int k=sum();

System.out.println("sum of two numbers is ="+k);

}

//callie method

public static int sum()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

//logic

int c = a+b;

return c;

}

}

Q) Write a java program to check given number is palindrome or not by using with return type with no argument method?

Input:

121

output:

It is a palindrome number

approach1

---------

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

String k=palindrome();

System.out.println(k);

}

//callie method

public static String palindrome()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int temp=n;

int rem,rev=0;

while(n>0)

{

rem=n%10;

rev=rev\*10+rem;

n=n/10;

}

if(temp==rev)

return "It is a palindrome number";

else

return "It is not a palindrome number";

}

}

approach2

----------

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

boolean k=palindrome();

if(k)

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

//callie method

public static booleanpalindrome()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int temp=n;

int rem,rev=0;

while(n>0)

{

rem=n%10;

rev=rev\*10+rem;

n=n/10;

}

if(temp==rev)

return true;

else

return false;

}

}

approach3

---------

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

int k=palindrome();

if(k==1)

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

//callie method

public static int palindrome()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int temp=n;

int rem,rev=0;

while(n>0)

{

rem=n%10;

rev=rev\*10+rem;

n=n/10;

}

if(temp==rev)

return 1;

else

return 0;

}

}

4) With returntype with Argument method

---------------------------------------

Q) Write a java program to perform sum of two numbers using with returntype with argument method?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

//caller method

System.out.println("sum of two number is ="+sum(a,b));

}

//callie method

public static int sum(int a,int b)

{

int c = a + b;

return c;

}

}

Q) write a java program to check given number is palindrome or not using with returntype with argument method?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//caller method

if(palindrome(n))

System.out.println("It is palindrome number");

else

System.out.println("It is not palindrome number");

}

//callie method

public static booleanpalindrome(int n)

{

int temp=n;

int rem,rev=0;

while(n>0)

{

rem=n%10;

rev=rev\*10+rem;

n=n/10;

}

if(temp==rev)

return true;

else

return false;

}

}

Assignment

===========

Q) Write a java program to check given number is prime or not using 3 and 4th approach?

Recursion

===========

A method which call itself for many number of times is called recursion.

Recursion is similar toloopings.

Whenever we use recursion we should not use loops.

Q) Write a java program to display 10 natural numbers without using loops?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

display(1);

}

//callie method

public static void display(int i)

{

if(i<=10)

{

System.out.print(i+" "); //1 2 3 4 5 6 7 8 9 10

display(i+1);

}

}

}

Loop Pattern

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

Left Side Loop Patterns

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

Right side loop patterns

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

Pyramid loop patterns

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

Class =26

Arrays

=======

Array is a collection of homogeneous data elements.

The main advantages of arrays are

1) We can represent multiple elements using single variable name.

ex:

int[] arr={10,20,30};

2) Performance point of view arrays arerecommanded to use.

The main disadvantages of arrays are

1) It is fixed in the size.Once if we create an array there is no chance of increase or decreasing

the size of an array.

2) To use array concept in advanced we should know what is the size of array which is always not

possible.

In java, arrays are divided into three types .

1) Single Dimensional Array

2) Double Dimensional Array

3) Multi Dimensional Array

Array Declaration

-----------------

At the time of array declaration we should not specify array size.

Arrays

|----------------------------------|-------------------------------------|

Single Dimensional Array Double Dimensional Array Multi Dimensional Array

int[] arr; int[][] arr; int[][][] arr;

int []arr; int [][]arr; int [][][]arr;

int arr[]; int ar[][]; int arr[][][];

int[] []arr; int[][] []arr;

int[] arr[]; int[][] arr[];

int []arr[]; int[] [][]ar;

int[] arr[][];

int[] []arr[];

int [][]arr[];

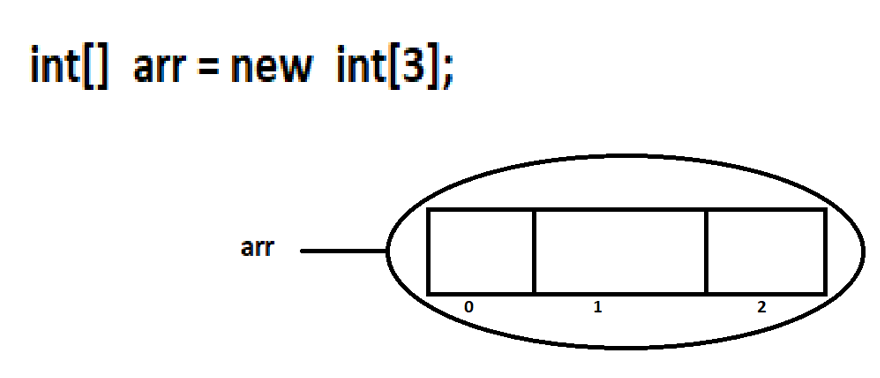
int []arr[][];

Array creation

--------------

In java, every array consider as an object.Hence we will use new operator to create an array.

Diagram: class26.1



Rules to construct an array:

----------------------------

Rule1:

-----

At the time of array creation compulsary we need to specify array size.

ex:

int[] arr=new int[3];

int[] arr=new int[]; // C.T.E array dimension missing

Rule2:

-----

It is legal to have an array size with zero.

ex:

int[] arr=new int[0];

System.out.println(arr.length);

Rule3:

-----

We can't take negative numbers as an array size otherwise we will get

NegativeArraySizeException.

ex:

int[] arr=new int[-3]; //R.E NegativeArraySizeException

Rule4:

-----

The allowed datatype for an array size is byte,short,int and char.

If we take other datatyps then we will get compile time error.

ex:

int[] arr=new int['a']; //97

byte b=10;

int[] arr=new int[b]; // 10

int[] arr=new int[10.5]; //invalid

Rule5:

-----

The maximum length we can give for an array size is maximum length of integer.

ex:

int[] arr=new int[2147483647];

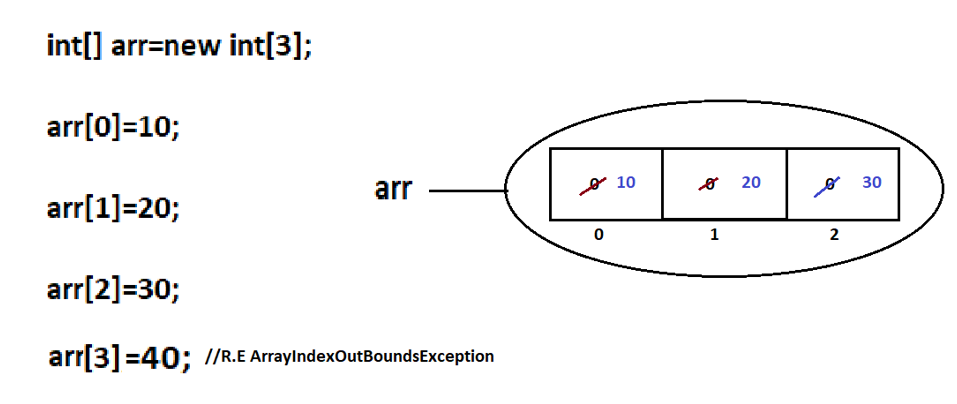
Array initialization

-------------------

Whenever we create an array , every array element will be initialized with default vlaues.

IF we are not happing with default values then we can change with customized values.

Diagram: class26.2



Array Declaration , Creation and Initialization using single line

------------------------------------------------------------------

int[] arr;

arr=new int[3];

arr[0]=10;

arr[1]=20;

arr[2]=30; ===> int[] arr={10,20,30};

===> char[] carr={'a','b','c'};

===> String[] sarr={"hi","hello","bye"};

Q) What is the difference between length and length() method?

length

------

It is a final variable which is applicable for arrays.

It will return size of an array.

ex:

class Test

{

public static void main(String[] args)

{

int[] arr=new int[3];

System.out.println(arr.length); //3

}

}

length()

--------

It is a final method which is applicable for String objects.

It will return number of characters present in String.

ex:

class Test

{

public static void main(String[] args)

{

String str="ihub";

System.out.println(str.length()); //4

}

}

Single Dimensional Array Programs

----------------------------------

Q) Write a java program to insert some elements in array and display them?

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the array size :");

int size=sc.nextInt(); //4

int[] arr=new int[size];

//insert elements

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

{

System.out.println("Enter the element :");

arr[i]=sc.nextInt();

}

//display elements

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

{

System.out.print(arr[i]+" ");

}

}

}

iv) for each loop

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

It is used to iterate the elements from array.

ex:

--

class Test

{

public static void main(String[] args)

{

int[] arr={10,20,30};

//for each loop

for(int i:arr)

{

System.out.print(i+" ");

}

}

}

**ARRAY METHODS**

1. **Arrays.sort(array):** Sorts the specified array into ascending order.
2. **Arrays.binarySearch(array, key):** Searches for the specified key in a sorted array and returns its index.
3. **Arrays.copyOf(array, newLength):** Creates a new array that is a copy of the specified array with the specified length.
4. **Arrays.copyOfRange(array, from, to):** Copies the specified range of the array into a new array.
5. **Arrays.fill(array, value):** Fills the specified array with the specified value.
6. **Arrays.equals(array1, array2):** Checks if two arrays are equal (contain the same elements in the same order).
7. **Arrays.toString(array):** Returns a string representation of the array.
8. **Arrays.deepToString(array)**: Returns a string representation of a multi-dimensional array.
9. **Arrays.asList(array):** Converts an array to a fixed-size list backed by the array.
10. **Arrays.stream(array):** Returns a sequential stream with the specified array as its source.

**String to char**

* **char charValue = myString.charAt(index);**: Retrieves the character at the specified index from the string.

**char to int**

* **int intValue = (int) myChar;:** Converts a character to its ASCII (or Unicode) integer value.

**int to String**

* **String strValue = Integer.toString(myInt);:** Converts an integer to its string representation.

**String to int**

* **int intValue = Integer.parseInt(myString);:** Converts a string representation of an integer to an actual integer.

**String to char array**

* **char[] charArray = myString.toCharArray();:** Converts the string into an array of characters.

**char array to String**

* **String strValue = new String(charArray);:** Creates a new string from an array of characters.

class Test

{

public static void main(String[] args)

{

Integer i1=new Integer(10);

System.out.println(i1); //10

Integer i2=new Integer("20");

System.out.println(i2); //20

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

Boolean b1=new Boolean(true);

System.out.println(b1);

Boolean b2=new Boolean("false");

System.out.println(b2);

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

Character c=new Character('a');

System.out.println(c);//a

}

}

Class 30

Double Dimensional Array

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

Double dimensional array is a combination of rows and columns.

Double dimensional array is implemented based on array or arrays approach but not matrix form.

The main objective of double dimensional array is memory utilization.

Double dimensional array is used to develop business oriented application, gaming applications, matrix type of applications and etc.

We can declare dimensional array as follow.

ex:

columns

|

int[][] arr=new int[3][3];

|

rows

Here we can store 9 elements.

Anonymous Array

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

Sometimes we will declare an array without name such type of nameless array is called anonymous array.

The main objective of anonymous array is just for instance use.

We can declare anonymous array as follow.

ex:

new int[]{10,20,30};

new int[][]{{1,2,3},{4,5,6}};

Interview Question

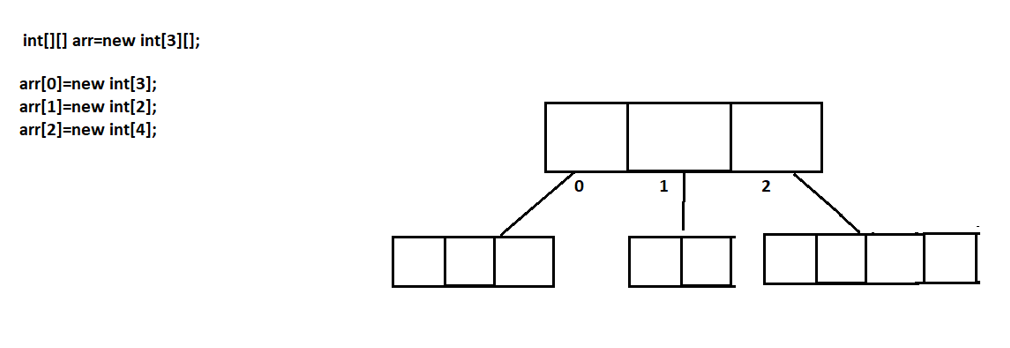
===================

Q) What is jagged array?

Jagged array is also known as array of arrays.

It is a multi dimensional array where each row having different size.

Diagram: class33.4



----------------------------------------------------------------------------

////////////////////////////////////////////////////////////////////////////////

----------------------------------------------------------------------------

OOPSS CONCEPT FROM CLASS 30-36

----------------------------------------------------------------------------

////////////////////////////////////////////////////////////////////////////////

----------------------------------------------------------------------------

Class 37

Package

========

A package is a collection of classes,interfaces,enums and annotations.

Enum is a special class and annotation is a special interface.

In general , a package is a collection of classes and interfaces.

A package is also known as folder or a directory.

In java packages are divided into two types.

1) Predefined packages

--------------------

Built-In packages are called predefined packages.

ex:

java.lang

java.io

java.util

java.util.stream

java.time

and etc.

2) Userdefined packages

-----------------------

A package which is created by the user based on the application requirement is called predefined package.

To declare predefined package we need to use package keyword.

It is recommanded to declare a package name in the reverse order of url.

syntax:

-----

package <package\_name>;

ex:

package com.google.www;

ex:

----

package com.ihub.www;

import java.util.Calendar;

class Test

{

public static void main(String[] args)

{

Calendar c=Calendar.getInstance();

int h=c.get(Calendar.HOUR\_OF\_DAY);

if(h<12)

System.out.println("Good Morning");

else if(h<16)

System.out.println("Good Afternoon");

else if(h<20)

System.out.println("Good Evening");

else

System.out.println("Good Night");

}

}

We can compile above program by using below command.

ex:

current directory

|

javac -d . Test.java

|

destination folder

We can run above program by using below command.

ex:

java com.ihub.www.Test

|

package name

Enum

=====

Enum is a group of named constants.

Enum concept introduced in 1.5 version.

Using enum we can create our own datatype called enumerated datatype.

When compare to old language enum, java enum is more powerful.

To declare the enum we will use enum keyword.

syntax:

------

enum type\_name

{

value1,value2,.....,valueN

}

Internal implementation of enum

---------------------------------

Every enum internally implements as class concept and it extends with java.lang.Enum class.

Every enum constant is a reference variable of enum type.

ex:

enum Months final class Months extends java.lang.Enum

{ {

JAN,FEB,MAR ==> public static final Months JAN=new Months();

} public static final Months FEB=new Months();

public static final Months MAR=new Months();

}

Declaration and Usage of enum

--------------------------------

enum Months

{

JAN,FEB,MAR

}

class Test

{

public static void main(String[] args)

{

Months m=Months.FEB;

System.out.println(m);//FEB

}

}

ex:

---

enum Months

{

JAN,FEB,MAR

}

class Test

{

public static void main(String[] args)

{

Months m=Months.FEB;

switch(m)

{

case JAN: System.out.println("January"); break;

case FEB: System.out.println("February"); break;

case MAR: System.out.println("March"); break;

}

}

}

java.lang.Enum class

--------------------

The power to enum will be inherited from java.lang.Enum class.

It contains following two methods.

1) values()

---------

It is used to read set of constants from enum.

2) ordinal()

-----------

It is used to display ordinal number.

ex:

---

enum Week

{

MON,TUE,WED,THU,FRI,SAT,SUN

}

class Test

{

public static void main(String[] args)

{

Week[] w=Week.values();

for(Week w1:w)

{

System.out.println(w1+" ------------- "+w1.ordinal());

}

}

}

When compare to old language enum, java enum is more powerful because in addition to constants

we can declare variables, methods and constructors.

ex:

----

enum Cloths

{

SILK,COTTON,KHADI;

Cloths()

{

System.out.println("constructor");

}

}

class Test

{

public static void main(String[] args)

{

Cloths c=Cloths.SILK;

}

}

ex:

---

enum Cloths

{

SILK,COTTON,KHADI;

static int i=100;

public static void main(String[] args)

{

System.out.println(i);

}

}

Singleton Class

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

Singleton is one of the design pattern which allows us to create only one object for a class.

Using a class name if we call any method and that method returns same class object is called singleton class.

ex:

Calendar c = Calendar.getInstance();

LocalDate date = LocalDate.now();

LocalTime time = LocalTime.now();

To create a singleton class we required private constructor and static method.

ex:

---

class Singleton

{

static Singleton singleton=null;

private Singleton()

{

}

public static Singleton getInstance()

{

if(singleton==null)

{

singleton=new Singleton();

}

return singleton;

}

}

class Test

{

public static void main(String[] args)

{

Singleton s1=Singleton.getInstance();

System.out.println(s1.hashCode());

Singleton s2=Singleton.getInstance();

System.out.println(s2.hashCode());

Singleton s3=Singleton.getInstance();

System.out.println(s3.hashCode());

}

}

Wrapper classes

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

The main objective of wrapper classes are.

1) To wrap primitive type to wrapper object and vice versa.

2) To define several utility method.

ex:

Primitive type wrapper class

----------------- -------------

byte Byte

short Short

int Integer

long Long

float Float

double Double

boolean Boolean

char Character

constructor

------------

There are two ways to create object for wrapper classes. One will take corresponding primitive as an argument and another will take corresponding String as an argument.

ex:

wrapper class constructor

----------------- -------------

Byte byte or String

Short short or String

Integer int or String

Long long or String

Float float or String

Double double or String

Boolean boolean or String

Character char

Utility methods

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

1) parseXxx()

--------------

It is used to convert string to primitive type.

ex:

class Test

{

public static void main(String[] args)

{

String str="10";

int i= Integer.parseInt(str);

System.out.println(i); // 10

long l= Long.parseLong(str);

System.out.println(l); // 10

float f= Float.parseFloat(str);

System.out.println(f); // 10.0

double d= Double.parseDouble(str);

System.out.println(d); // 10.0

}

}

2) toString()

----------------

It is used to convert wrapper object to string.

ex:

--

class Test

{

public static void main(String[] args)

{

Integer i1=new Integer(10);

String str =i1.toString();

System.out.println(str); // 10

}

}

3) xxxValue()

---------------

It is used to convert wrapper object to primitive type.

ex:

class Test

{

public static void main(String[] args)

{

Integer i1=new Integer(10);

byte b = i1.byteValue();

System.out.println(b);

short s = i1.shortValue();

System.out.println(s);

}

}

Class 38

**Inner classes**

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

Sometimes we will declare a class inside another class such concept is called inner class.

ex:

class Outer\_class

{

class Inner\_class

{

-

-

-

}

}

Inner classes introduced as a part of event handling to remove GUI bugs.

But due to powerful features and benefits of inner classes. Programmers started to use inner classes in regular programming.

Inner class does not allow static members.

Accessing inner class data from static area of outer class

----------------------------------------------------------

class Outer

{

class Inner

{

public void m1()

{

System.out.println("Inner-M1 Method");

}

}

public static void main(String[] args)

{

Outer.Inneri=new Outer().new Inner();

i.m1();

}

}

If we compile above program we will get two .class files i.eOuter.class and Outer$Inner.clas.

ex:

----

class Outer

{

class Inner

{

public void m1()

{

System.out.println("Inner-M1 Method");

}

}

public static void main(String[] args)

{

new Outer().new Inner().m1();

}

}

Accessing inner class data from non-static area of outer class

---------------------------------------------------------------

class Outer

{

class Inner

{

}

public void m1()

{

System.out.println("Inner-M1 Method");

}

}

public void m2()

{

Inner i=new Inner();

i.m1();

}

public static void main(String[] args)

{

Outer o=new Outer();

o.m2();

}

}

class Outer

{

class Inner

{

public static void m1()

{

System.out.println("Inner-M1 Method");

}

}

public void m2()

{

Inner i=new Inner();

i.m1();

}

public static void main(String[] args)

{

Outer o=new Outer();

o.m2();

}

}

o/p:

C.T.E : Illegal static declaration in inner class

Types of objects in java

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

We have two types of objects in java.

1) Immutable object

2) Mutable object

1) Immutable object

--------------------

After object creation if we perform any changes then for every change a new object is will be created such type of object is called immutable object.

ex:

String and Wrapper classes

2) Mutable object

-----------------

After object creation if we perform any changes then all the required changes will be done in a same object such type of object is called mutable object.

ex:

StringBuffer and StringBuilder

String

======

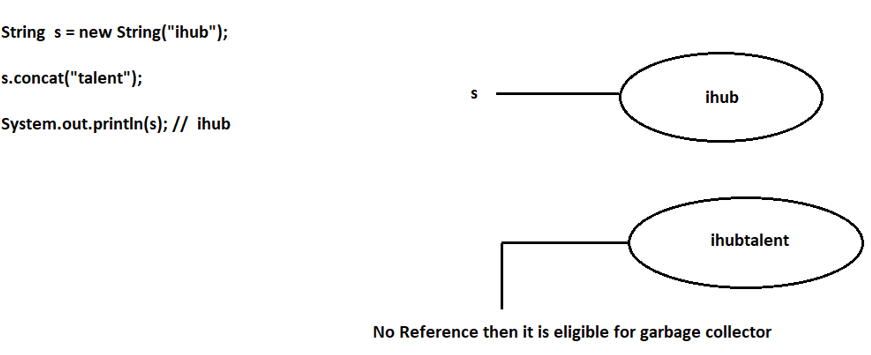
It is a collection of characters which is enclosed in a double quotation.

case1:

------

Once if we create a String object we can't perform any changes.If we perform any changes then for every change a new object will be created such behaviour is called immutability of an object.

Diagram: class38.1



**case2:**

**-----**

What is the difference between == and .equals() method?

**==**

**----**

It is a comparision operator which returns boolean value either true or false.

It is used for reference comparision or address comparision.

ex:

class Test

{

public static void main(String[] args)

{

String s1=new String("ihub");

String s2=new String("ihub");

System.out.println(s1==s2); // false

}

}

**.equals()**

**----------**

It is a predefined method present in String which returns boolean value either true or false.

It is used for content comparision and it is a case sensitive.

ex:

---

class Test

{

public static void main(String[] args)

{

String s1=new String("ihub");

String s2=new String("ihub");

System.out.println(s1.equals(s2)); // true

}

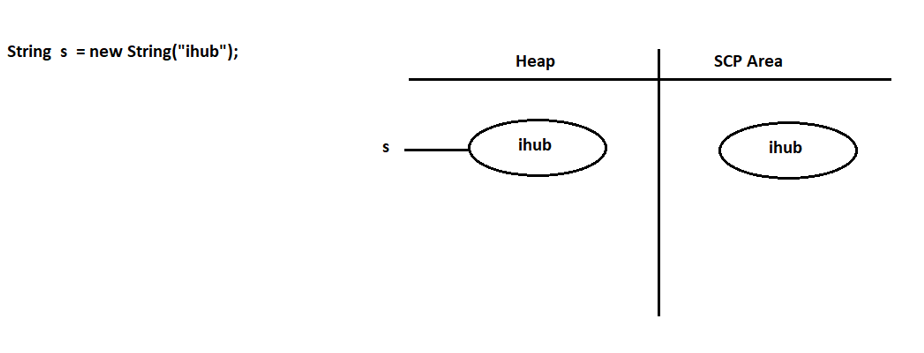
}

**case3:**

**-----**

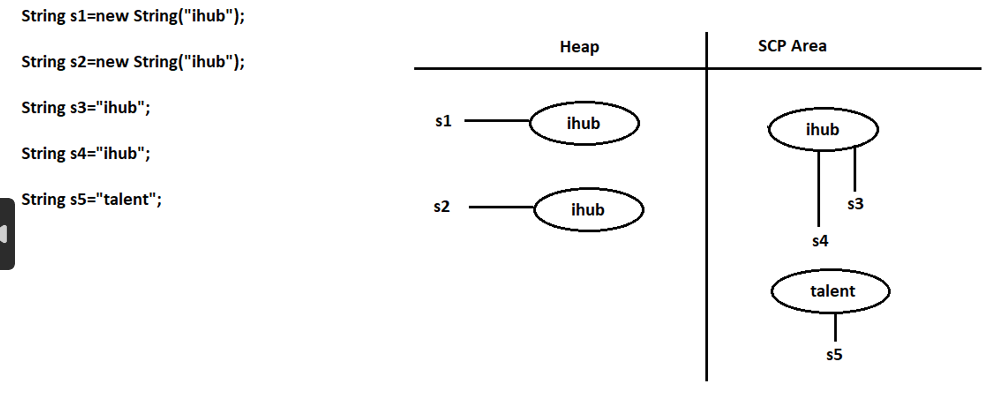
Once if we create a String object.Two objects will be created one is on heap and another is on SCP area.But 's' always points to heap area.

**Diagram: class38.2**



* Object creation in SCP area is always optional.First JVM will check is there object is created with same content or not. If it is not created then JVM will create a new object.If it is created then JVM Won't create any new object.Hence there is no chance of having duplicate objects in SCP area.
* SCP area objects do not have any object reference even though garbage collector can't access them.
* SCP objects will destroy automatically when JVM shutdowns or terminated.

**Diagram: class38.3**

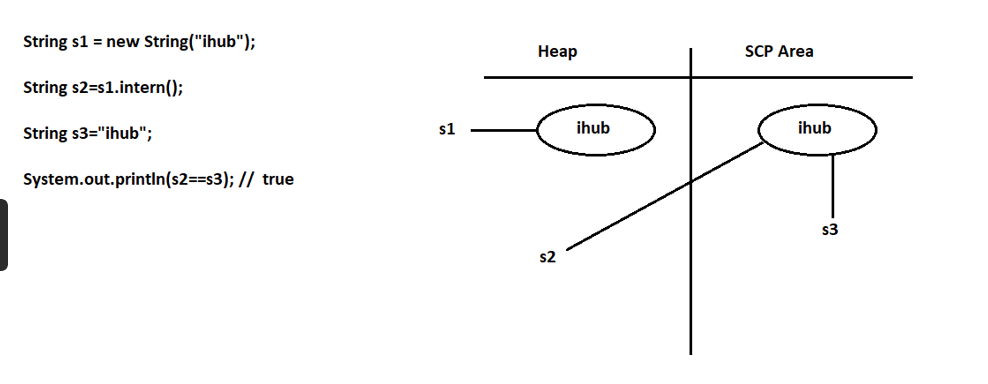


**Interning of String object**

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

With the help of heap object reference if we need corresponding SCP object reference then we need to use intern() method.

**Diagram: class38.4**



String important methods

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

Q) Write a java program to find out length of the string?

input:

hello

output:

5

ex:

class Test

{

public static void main(String[] args)

{

String str="hello";

System.out.println(str.length());

}

}

Q) Write a java program to display string character by character?

input:

hello

output:

h

e

l

l

o

ex:

class Test

{

public static void main(String[] args)

{

String str="hello";

for(int i=0;i<str.length();i++)

{

System.out.println(str.charAt(i));

}

}

}

Q) Write a java program to concatinate two strings?

input:

ihub

talent

output:

ihubtalent

ex:

---

class Test

{

public static void main(String[] args)

{

String str1="ihub";

String str2="talent";

System.out.println(str1.concat(str2));

}

}

Q) Write a java program to check both strings are equal or not?

input:

ihub

ihub

output:

Both are equals

ex:

class Test

{

public static void main(String[] args)

{

String str1="ihub";

String str2="ihub";

if(str1.equals(str2))

System.out.println("Both are equals");

else

System.out.println("Both are not equals");

}

}

Q) Write a java program to check both strings are equal or not?

input:

IHUB

ihub

output:

Both are equals

ex:

class Test

{

public static void main(String[] args)

{

String str1="IHUB";

String str2="ihub";

if(str1.equalsIgnoreCase(str2))

System.out.println("Both are equals");

else

System.out.println("Both are not equals");

}

}

Q) Write a java program to convert uppercase string to lowercase ?

input:

IHUBTALENT

output:

ihubtalent

ex:

class Test

{

public static void main(String[] args)

{

String str="IHUBTALENT";

str=str.toLowerCase();

System.out.println(str);

}

}

Q) Write a java program to convert lowercase string to uppercase ?

input:

ihubtalent

output:

IHUBTALENT

ex:

class Test

{

public static void main(String[] args)

{

String str="ihubtalent";

str=str.toUpperCase();

System.out.println(str);

}

}

Q) Write a java program to remove special characters from given string?

input:

I\_h@ub$Tale#nt1

output:

IhubTalent1

ex:

---

class Test

{

public static void main(String[] args)

{

String str="I\_h@ub$Tale#nt1";

str=str.replaceAll("[^A-Za-z0-9]","");

System.out.println(str);

}

}

Q) Write a java program to remove the spaces from string?

input:

I hub Tale nt

output:

IhubTalent

ex:

---

class Test

{

public static void main(String[] args)

{

String str="I hub Tale nt";

str=str.replaceAll("\\s","");

System.out.println(str);

}

}

Q) Write a java program to concatinate two strings?

input:

Ihub12

Talent18

output:

IhubTalent30

ex:

---

class Test

{

public static void main(String[] args)

{

String str1="Ihub12";

String str2="Talent18";

String word1=str1.replaceAll("[^A-Za-z]","");

int num1=Integer.parseInt(str1.replaceAll("[^0-9]",""));

String word2=str2.replaceAll("[^A-Za-z]","");

int num2=Integer.parseInt(str2.replaceAll("[^0-9]",""));

String word = word1 + word2;

int num = num1 + num2;

System.out.println(word+num);

}

}

Q) Write a java program to display the string in reverse order?

input:

hello

output:

olleh

class Test

{

public static void main(String[] args)

{

String str="hello";

for(int i=str.length()-1;i>=0;i--)

{

System.out.print(str.charAt(i));

}

}

}

Assignment

----------

Q) Write a java program to find out given string is palindrome or not?

input:

racar

output:

It is a palindrome string

Class 39

Q) Write a java program to display reverse of a string?

input:

hello

output:

olleh

ex:

---

class Test

{

public static void main(String[] args)

{

String str="hello";

char[] carr=str.toCharArray(); // h e l l o

//reverse variable

String rev="";

//reading reverse

for(int i=carr.length-1;i>=0;i--)

{

rev+=carr[i];

}

System.out.println(rev);

}

}

Q) Write a java program to check given string is palindrome or not?

input:

racar

output:

It is a palindrome string

ex:

---

class Test

{

public static void main(String[] args)

{

String str="racar";

char[] carr=str.toCharArray(); //r a c a r

//reverse variable

String rev="";

//reading reverse

for(int i=carr.length-1;i>=0;i--)

{

rev+=carr[i];

}

if(str.equals(rev))

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

}

Q) Write a java program to display reverse of a sentence?

Input:

This is java class

output:

class java is This

ex:

class Test

{

public static void main(String[] args)

{

String str="This is java class";

String[] sarr=str.split(" "); // This is java class

//reading reverse

for(int i=sarr.length-1;i>=0;i--)

{

System.out.print(sarr[i]+" ");

}

}

}

Q) Write a java program to display reverse of a string?

input:

This is java class

output:

sihTsiavajssalc

ex:

---

class Test

{

public static void main(String[] args)

{

String str="This is java class";

String[] sarr=str.split(" "); // This is java class

//for each loop

for(String s:sarr)

{

char[] carr=s.toCharArray(); // T h i s

for(int i=carr.length-1;i>=0;i--)

{

System.out.print(carr[i]);

}

//space

System.out.print(" ");

}

}

}

Q) Write a java program to display the string starting with uppercase letter?

input:

This is Java class For students

output:

This Java For

ex:

class Test

{

public static void main(String[] args)

{

String str="This is Java class For students";

String[] sarr=str.split(" ");

//for each loop

for(String s:sarr)

{

char ch=s.charAt(0);

if(ch>='A' &&ch<='Z')

{

System.out.print(s+" ");

}

}

}

}

Q) Write a java program to check given string is anagram or not?

input:

silent

listen

output:

It is a anagram string

import java.util.Arrays;

class Test

{

public static void main(String[] args)

{

String str1="silent";

String str2="listen";

char[] carr1=str1.toCharArray();

char[] carr2=str2.toCharArray();

Arrays.sort(carr1); // e i l n s t

Arrays.sort(carr2); // e i l n s t

boolean flag=true;

for(int i=0;i<carr1.length &&i<carr2.length;i++)

{

if(carr1[i]!=carr2[i])

{

flag=false;

break;

}

}

if(flag==true)

System.out.println("It is a Anagram string");

else

System.out.println("It is not a Anagram string");

}

}

Q) Write a java program to perform right rotation of a given string?

input:

str = ihubtalent

cnt = 4

output:

talentihub

ex

---

class Test

{

public static void main(String[] args)

{

String str="ihubtalent";

int cnt=4;

String str1=str.substring(0,cnt);

String str2=str.substring(cnt,str.length());

str=str2+str1;

System.out.println(str);

}

}

Q) Write a java program to get below below output?

input:

str = Ihubtalentstudents

index = 10

word = for

output:

Ihubtalentforstudents

ex:

---

class Test

{

public static void main(String[] args)

{

String str="Ihubtalentstudents";

int index=10;

String word="for";

String word1=str.substring(0,index);

String word2=str.substring(index,str.length());

str = word1+word+word2;

System.out.println(str);

}

}

Q) Write a java program to display the string in a given format?

input:

XYZ

output:

XY

XZ

YX

YZ

ZX

ZY

ex:

class Test

{

public static void main(String[] args)

{

String str="XYZ";

for(int i=0;i<str.length();i++)

{

for(int j=0;j<str.length();j++)

{

if(i!=j)

{

System.out.println(str.charAt(i)+""+str.charAt(j));

}

}

}

}

}

Assignment

===========

Q) Write a java program to display palindrome strings?

input:

racar is madam for java

output:

racar madam

Class 40

Q) Write a java program to display unique/distinct characters from given string?

input:

google

output:

gole

ex:

----

class Test

{

public static void main(String[] args)

{

String str="google";

String duplicates="";

String unique="";

for(int i=0;i<str.length();i++)

{

String current=Character.toString(str.charAt(i));

if(unique.contains(current))

{

if(!duplicates.contains(current))

{

duplicates+=current;

continue;

}

}

unique+=current;

}

System.out.println(unique);

}

}

Q) Write a java program to display duplicate characters from given string?

input:

google

output:

og

ex:

---

class Test

{

public static void main(String[] args)

{

String str="google";

String duplicates="";

String unique="";

for(int i=0;i<str.length();i++)

{

String current=Character.toString(str.charAt(i));

if(unique.contains(current))

{

if(!duplicates.contains(current))

{

duplicates+=current;

continue;

}

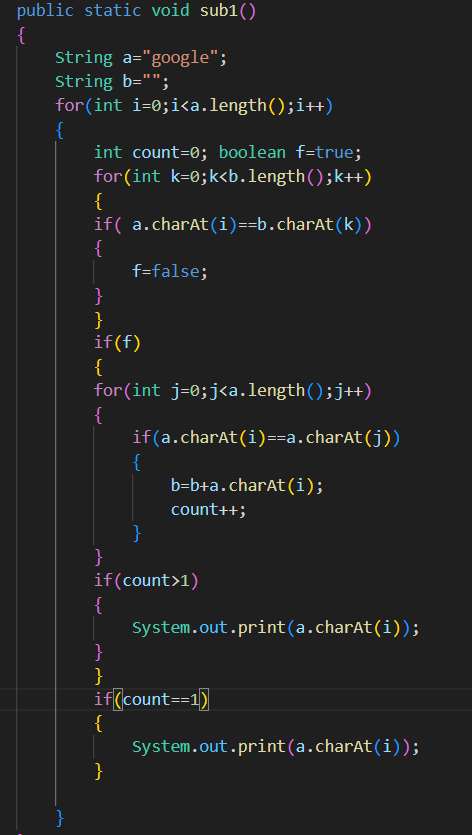
}

unique+=current;

}

System.out.println(duplicates);

}

}

Q) Write a java program to display most repeating character in a given string?

input:

ihubtalentinstitute

output:

t repeating for 5 times

ex:

---

class Test

{

public static void main(String[] args)

{

String str="ihubtalentinstitute";

char character=' ';

int maxCount=0;

for(int i=0;i<str.length();i++)

{

int cnt=0;

for(int j=0;j<str.length();j++)

{

if(str.charAt(i) == str.charAt(j))

{

cnt++;

}

}

if(maxCount<cnt)

{

maxCount=cnt;

character=str.charAt(i);

}

}

System.out.println(character+" repeating for "+maxCount+" times");

}

}

Q) Write a java program to display the string in a given format?

input:

A1B2C3D4

output:

ABBCCCDDDD

ex:

class Test

{

public static void main(String[] args)

{

String str="A1B2C3D4";

for(int i=0;i<str.length();i++)

{

if(Character.isAlphabetic(str.charAt(i)))

{

System.out.print(str.charAt(i));

}

else

{

int n=Character.getNumericValue(str.charAt(i));

for(int k=1;k<n;k++)

{

System.out.print(str.charAt(i-1));

}

}

}

}

}

Q) Write a java program to perform permutation of a given string?

input:

ABC

output:

ABC

ACB

BAC

BCA

CBA

CAB

ex:

---

class Test

{

public static void main(String[] args)

{

String str="ABC";

permutation(str.toCharArray(),0);

}

public static void permutation(char[] arr,int fi)

{

if(fi==arr.length-1)

{

System.out.println(arr);

return;

}

for(int i=fi;i<arr.length;i++)

{

swapping(arr,fi,i);

permutation(arr,fi+1);

swapping(arr,fi,i);

}

}

public static void swapping(char[] arr,intfi,inti)

{

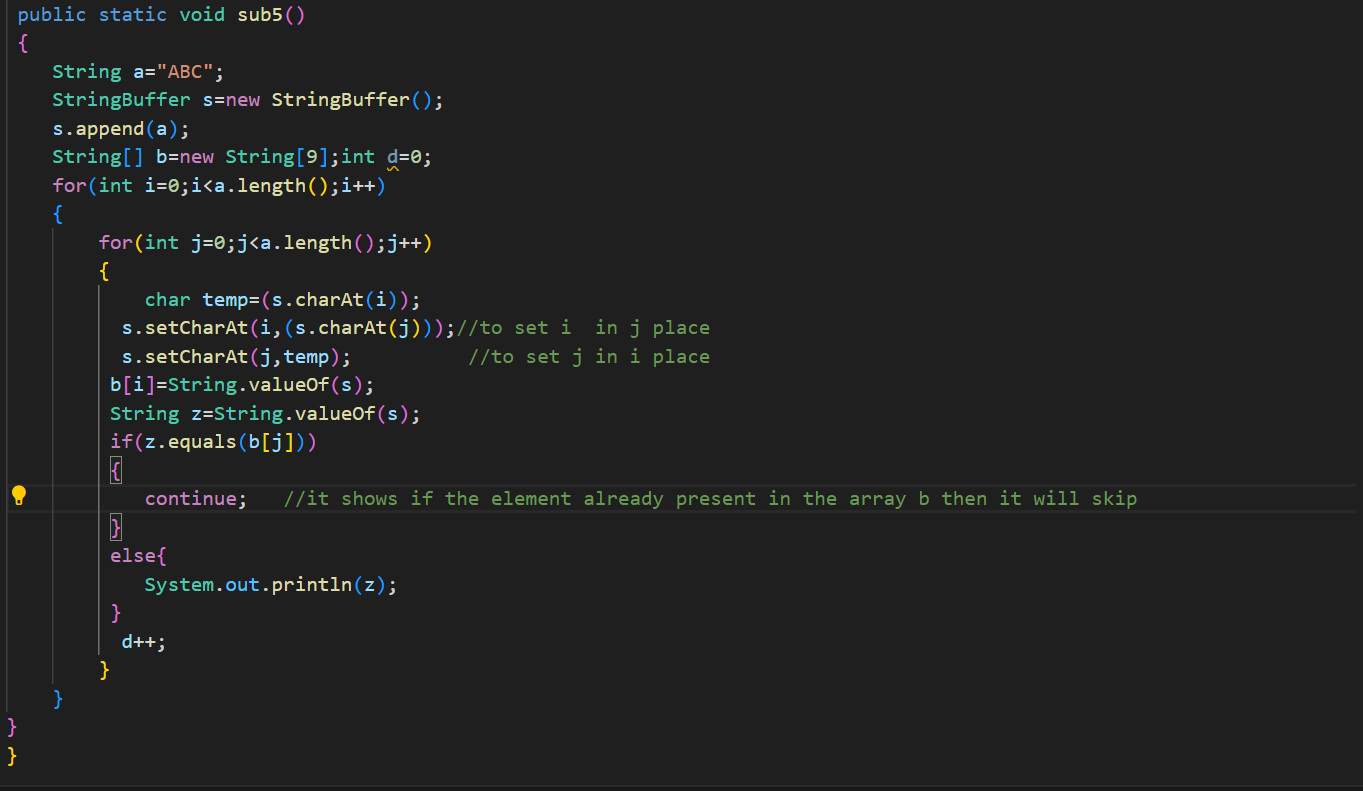
char temp=arr[fi];

arr[fi]=arr[i];

arr[i]=temp;

}

}



Q) Write a java program to encode the string?

input:

1106

output:

AAJF

class Test

{

public static void main(String[] args)

{

String str="1106";

for(int i=0;i<str.length();i++)

{

int n=Character.getNumericValue(str.charAt(i));

if(n>0)

{

System.out.print((char)('A'+n-1));

}

else

{

int k=Integer.parseInt(str.substring(i-1,i+1));

System.out.print((char)('A'+k-1));

}

}

}

}

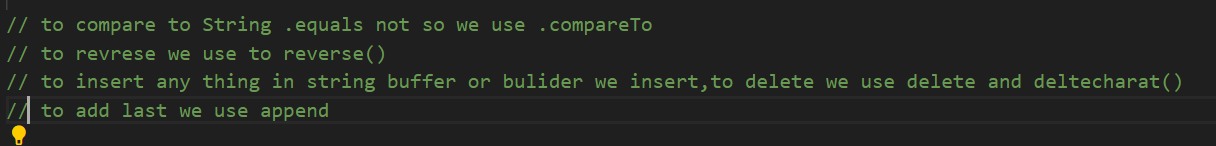
StringBuffer

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

If our content change frequently then it is never recommanded to use String because for every change a new object will be created.

To overcome this limitation Sun Micro System introduced StringBuffer concept.

In StringBuffer all the changes will reflect to single or same object only.



Methods used in Strings

**toLowerCase()**: Converts all characters in the string to lowercase.

**toUpperCase()**: Converts all characters in the string to uppercase.

**trim()**: Removes leading and trailing whitespace from the string.

**split(String regex)**: Splits the string into an array based on the specified regular expression.

**join(CharSequence delimiter, Iterable<? extends CharSequence> elements)**: Joins elements of an iterable into a single string, using the specified delimiter.

**replace(CharSequenceoldChar, CharSequencenewChar)**: Replaces occurrences of a substring with another substring.

**indexOf(String str)**: Returns the index of the first occurrence of the specified substring, or -1 if not found.

**charAt(int index)**: Returns the character at the specified index.

**substring(int beginIndex, int endIndex)**: Returns a substring from the specified begin index to the end index.

**startsWith(String prefix)**: Checks if the string starts with the specified prefix.

**endsWith(String suffix)**: Checks if the string ends with the specified suffix.

**contains(CharSequence sequence)**: Checks if the string contains the specified sequence of characters.

**length()**: Returns the number of characters in the string.

**isEmpty()**: Checks if the string is empty (length is 0).

**equals(Object anObject)**: Compares the string to the specified object for equality

Methods used in buffer and builder

**append(String str)**: Adds the specified string to the end of the buffer.

**insert(int offset, String str)**: Inserts the specified string at the given offset.

**delete(int start, int end)**: Removes the characters from the buffer between the specified start and end indices.

**replace(int start, int end, String str)**: Replaces the characters in the specified range with the given string.

**reverse()**: Reverses the sequence of characters in the buffer.

**capacity()**: Returns the current capacity of the buffer.

**length()**: Returns the number of characters currently stored in the buffer.

**substring(int start, int end)**: Returns a new string that contains the characters from the buffer between the specified start and end indices.

**setCharAt(int index, char ch)**: Replaces the character at the specified index with the. specified character

**toString()**: Converts the buffer to a string representation.

Methods used in buffer and builder

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

 **append(String str)**: Adds the specified string to the end of the buffer.

 **insert(int offset, String str)**: Inserts the specified string at the given offset.

 **delete(int start, int end)**: Removes the characters from the buffer between the specified start and end indices.

 **replace(int start, int end, String str)**: Replaces the characters in the specified range with the given string.

 **reverse()**: Reverses the sequence of characters in the buffer.

 **capacity()**: Returns the current capacity of the buffer.

 **length()**: Returns the number of characters currently stored in the buffer.

 **substring(int start, int end)**: Returns a new string that contains the characters from the buffer between the specified start and end indices.

 **setCharAt(int index, char ch)**: Replaces the character at the specified index with the. specified character

 **toString()**: Converts the buffer to a string representation.

**constructor**

**-----------**

1) StringBuffer sb=new StringBuffer()

-------------------------------------

It will create empty StringBuffer object with default initial capacity of 16.

If capacity reaches to maximum capacity then new capacity will be created with below formulea.

ex:

new capacity = current\_capacity + 1 \* 2 ;

ex:

----

class Test

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer();

System.out.println(sb.capacity()); // 16

sb.append("abcdefghijklmnop");

System.out.println(sb.capacity()); // 16

sb.append("qr");

System.out.println(sb.capacity()); // 16+1\*2=34

}

}

2) StringBuffer sb=new StringBuffer(int initialcapacity)

-----------------------------------------------

It will create StringBuffer object with specified initial capacity.

ex:

--

class Test

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer(19);

System.out.println(sb.capacity()); // 19

}

}

3) StringBuffer sb=new StringBuffer(String s)

-----------------------------------------------

It will create StringBuffer object equivalent to String.

Here capacity will be created with below formulea.

ex:

capacity = s.length() + 16

ex:

---

class Test

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("ihub");

System.out.println(sb.capacity()); // 4 + 16 = 20

}

}

Q) Write a java program to display reverse of a string?

input:

hello

output:

olleh

ex:

---

class Test

{

public static void main(String[] args)

{

String str="hello";

StringBuffer sb=new StringBuffer(str);

String rev=sb.reverse().toString();

System.out.println(rev);

}

}

Q) Write a java program to check given string is palindrome or not?

input:

racar

output:

It is a palindrome string

ex:

class Test

{

public static void main(String[] args)

{

String str="racar";

StringBuffer sb=new StringBuffer(str);

String rev=sb.reverse().toString();

if(str.equals(rev))

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

}

Assignment

===========

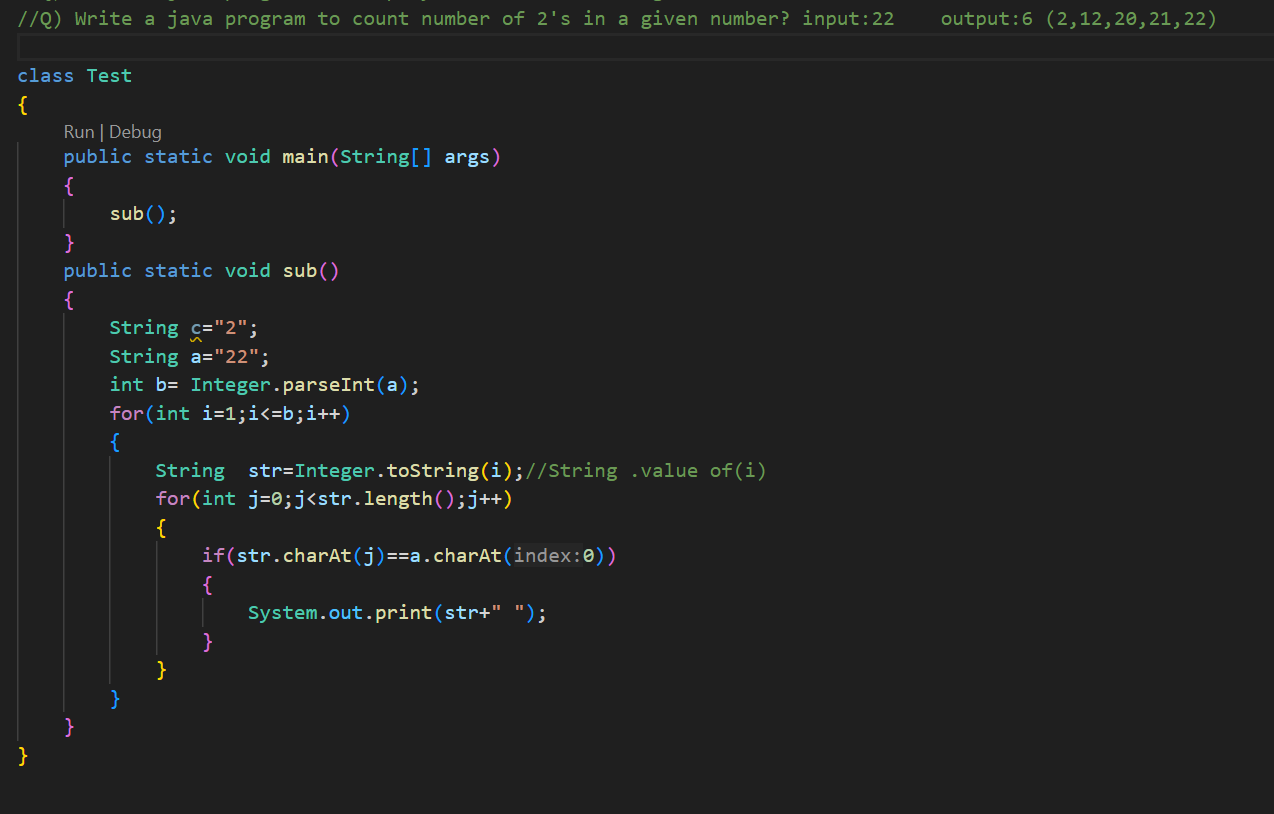
Q) Write a java program to count number of 2's in a given number?

input:

22

output:

6 (2,12,20,21,22)



Class 41

Q) Write a java program to count number of 2's present in a given number?

input:

22

output:

6 (2,12,20,21,22)

ex:

---

class Test

{

public static void main(String[] args)

{

int num=22;

StringBuffer sb=new StringBuffer();

for(int i=1;i<=num;i++)

{

sb.append(i);

}

int cnt=0;

for(int i=0;i<sb.length();i++)

{

int n=Character.getNumericValue(sb.charAt(i));

if(n==2)

{

cnt++;

}

}

System.out.println(cnt);

}

}

Q) Write a java program to multiply two arrays?

input:

5 3 1

2 5

output:

13275 (531\*25)

ex:

---

class Test

{

public static void main(String[] args)

{

int[] arr1={5,3,1};

int[] arr2={2,5};

//caller method

int a =Integer.parseInt(arrayToString(arr1));

int b =Integer.parseInt(arrayToString(arr2));

System.out.println(a\*b);

}

//callie method

public static String arrayToString(int[] arr)

{

StringBuffer sb=new StringBuffer();

for(int i:arr)

{

sb.append(i);

}

return sb.toString();

}

}

Q) Write a java program to display the string in a given format?

input:

ABBCCCDDDD

output:

A1B2C3D4

ex:

---

class Test

{

public static void main(String[] args)

{

String str="ABBCCCDDDD";

StringBuffer sb=new StringBuffer();

int count=1;

for(int i=0;i<str.length();i++)

{

if(i<str.length()-1 &&str.charAt(i) == str.charAt(i+1))

{

count++;

}

else

{

sb.append(str.charAt(i)).append(count);

count=1;

}

}

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

}

}

StringBuilder

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

StringBuilder is exactly same as StringBuffer with following differences.

StringBuffer StringBuilder

------------- --------------

Every method present in StringBuffer is No method present in StringBuilder is synchronized.

synchronized.

At a time only one thread is allowed to Multiple threads are allowed to operate on

operator on StringBufferobject.Hence it StringBuilder object.Hence it is not thread safe.

is thread safe.

Waiting time of a thread will increase There is no waiting threads relatively performance

relatively performance is low. is high.

It is introduced in 1.0v. It is introduced in 1.5v.

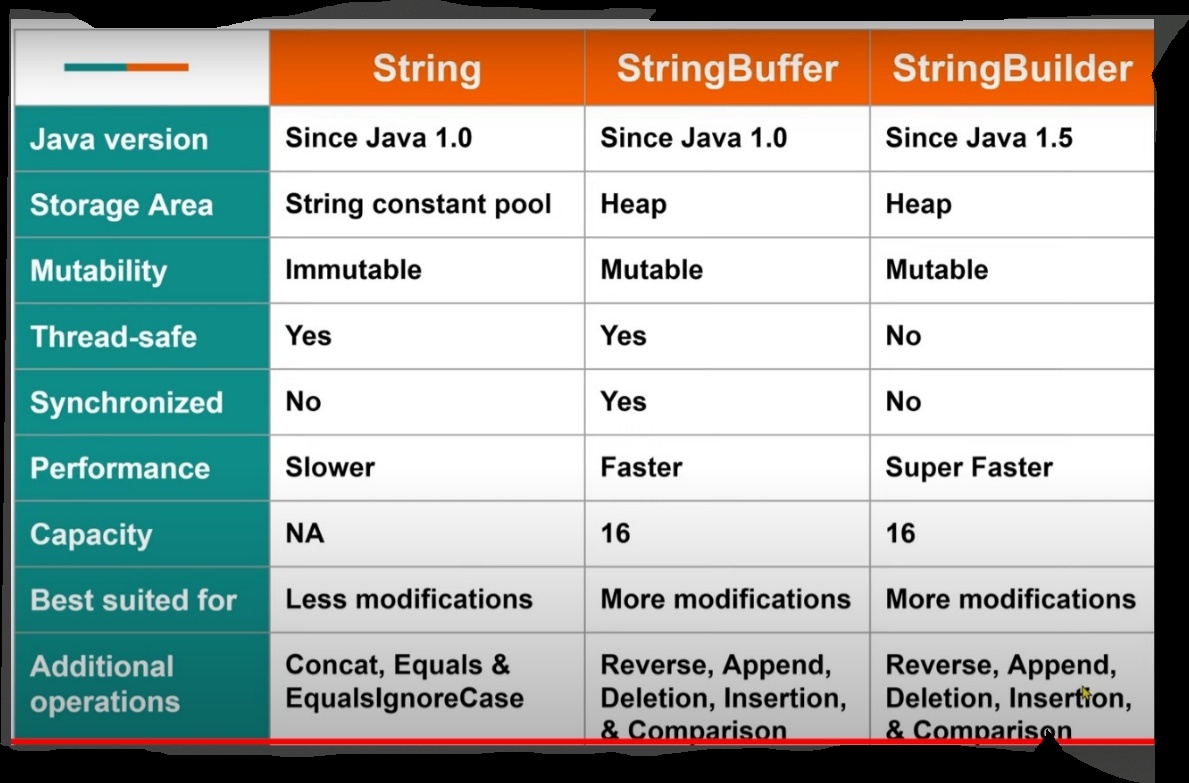
Note:

-----

If our content not change frequently then it is recommanded to use String.

If our content change frequently where thread safety is required then we need to use StringBuffer.

If our content change frequently where thread safety is not required then we need to use StringBuilder.



StringTokenizer

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

StringTokenizer is a class which is present in java.util package.

It is used to tokenize the string irrespective of regular expression.

We can create StringTokenizer class object as follow.

ex:

StringTokenizerst=new StringTokenizer(String s,RegularExpression regex);

StringTokenizer class contains following five methods.

ex:

public int countTokens()

public booleanhasMoreTokens()

public String nextToken()

public booleanhasMoreElements()

public Object nextElement()

ex:

---

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizerst=new StringTokenizer("this is java class"," ");

System.out.println(st.countTokens());//4

}

}

ex:

--

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizerst=new StringTokenizer("this is java class"," ");

while(st.hasMoreTokens())

{

String s=st.nextToken();

System.out.println(s);

}

}

}

ex:

---

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizerst=new StringTokenizer("this is java class"," ");

while(st.hasMoreElements())

{

String s=(String)st.nextElement();

System.out.println(s);

}

}

}

ex:

---

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizerst=new StringTokenizer("9,99,999",",");

while(st.hasMoreElements())

{

String s=(String)st.nextElement();

System.out.println(s);

}

}

}

Q) What is garbage collection and in how many ways we can call garbage collector?

Garbage is a daemon thread which is used to destroy unused or useless objects from java.

There are two ways to call garbage collector in java.

1) System.gc()

2) Runtime.getRuntime().gc()

ex:

--

class Test

{

//instance variable

int i=10;

public static void main(String[] args)

{

Test t1=new Test();

System.out.println(t1.i);//10

t1=null;

//System.gc();

Runtime.getRuntime().gc();

}

public void finalize()

{

System.out.println("Garbage Collector called");

}

}

Exception Handling

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

Q) What is the difference between Exception and Error?

Exception

---------

Exception is a problem for which we can provide solution programmatically.

Exceptions raised due to syntax errors.

ex:

ArithmeticException

FileNotFoundException

IllegalArgumentException

Error

-----

Error is a problem for which we can't provide solution programmatically.

Errors raised due to lack of system resources.

ex:

OutOfMemoryError

StackOverFlowError

LinkageError

As a part of java application development it is a responsibility of a programmer to provide smooth termination for every java program.

We have two types of terminations.

1) Smooth termination / Graceful termination

2) Abnormal termination

1) Smooth termination

---------------------

During the program execution suppose if we are not getting any interruption in the middle of the program such type of termination is called smooth termination.

ex:

class Test

{

public static void main(String[] args)

{

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

}

}

2) Abnormal termination

-----------------------

During the program execution suppose if we are getting any interruption in the middle of the program such type of termination is called abnormal termination.

ex:

class Test

{

public static void main(String[] args)

{

System.out.println(10/0);

}

}

If any exception raised in our program, we must and should handle that exception otherwise our program will terminates abnormally.

Here exception will display name of the exception, description of the exception and line number of the exception.

Exception

=========

It is a unwanted, unexpected event which disturbs normal flow of a program.

Exceptions always raise at runtime so they are also known as runtime events.

The main objective of exception handling is to provide graceful/smooth termination.

In java , exception are divided into two types.

1) Predefined exceptions

2) Userdefined exceptions

1) Predefined exceptions

--------------------------

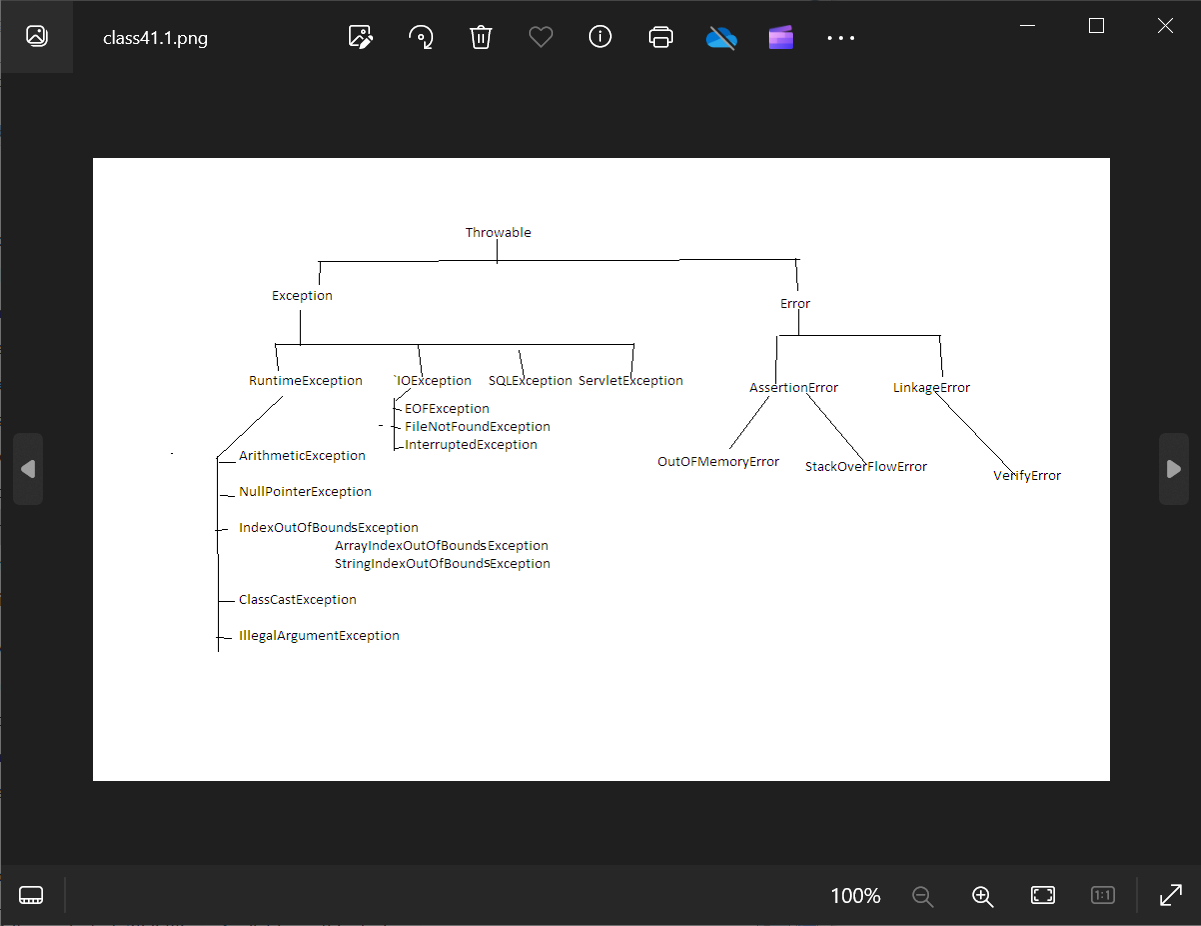
Built-In exceptions are called predefined exceptions.

It is categories into two types.

i) Checked exceptions

ii) Unchecked exceptions

Diagram; class41.1



i) Checked exceptions

-------------------

Exceptions which are checked by the compiler at the time of compilation are called checked exceptions.

ex:

InterruptedException

EOFException

IOException

ii) Unchecked exceptions

-------------------------

Exceptions which are checked by the JVM at the time of runtime are called unchecked exceptions.

ex:

ArithmeticException

ClassCastException

IllegalArgumentException

Class 42

If any checked exception raised in our program we must and should handle that exception by using try and catch block.

try block

==========

It is a block which contains risky code.

A try block associate with catch block.

If any exception raise in try block then it won't be executed.

A try block is used to throw the exception to catch block.

catch block

===========

It is a block which contains error handling code.

A catch block always associate with try block.

A catch block is used to catch the exception from try block.

If there is no exception in try block then catch block won't be executed.

A catch block takes exception name as a parameter and that name must match with exception class name.

syntax:

-----

try

{

-

- //Risky Code

-

}

catch(ArithmeticException ae)

{

-

-

-

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

catch(Exception e)

{

System.out.println("catch-block");

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(ArithmeticException ae)

{

System.out.println("catch-block");

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("stmt1");

System.out.println(10/0);

System.out.println("stmt2");

}

catch(ArithmeticException ae)

{

System.out.println("catch-block");

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

try

{

i++;

}

catch(Exception e)

{

i++;

}

System.out.println(i); // 11

}

}

A try with multiple catch blocks

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

A try block can have multiple catch blocks.

If a try block contains multiple catch blocks then order of catch block is very important it should be from child to parent but not from parent to child.

ex:

--

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch (ArithmeticException ae)

{

System.out.println("From AE");

}

catch (RuntimeException re)

{

System.out.println("From RE");

}

catch (Exception e)

{

System.out.println("From E");

}

}

}

Various methods to display exception details

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

Throwable class defines following methods to display exception details.

1) printStackTrace()

-----------------

It is used to display name of the exception, description of the exception and line number of the exception.

2) toString()

-----------

It is used to display name of the exception and description of the exception.

3) getMessage()

-------------

It is used to display description of the exception.

ex:

----

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(ArithmeticException ae)

{

ae.printStackTrace();

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

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

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

System.out.println(ae.getMessage());

}

}

}

Q) How to handle multiple exceptions in a single catch block?

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(IllegalArgumentException | ArithmeticException | ClassCastException e)

{

e.printStackTrace();

}

}

}

finally block

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

It is never recommanded to maintain cleanup code in try block because if any exception raised in try block then try block won't be executed.

It is never recommanded to maintain cleanup code in catch block because if there is no exception in try block then catch block won't be executed.

But we need a place where we can maintain cleanup code and it should execute irrespective of exception raised or not. Handle or not. Such block is called finally block.

syntax:

-----

try

{

-

- //Risky Code

-

}

catch(Exception e)

{

-

- //Error Handling Code

-

}

finally

{

-

- //Cleanup code

-

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

catch(Exception e)

{

e.printStackTrace();

}

finally

{

System.out.println("finally-block");

}

}

}

o/p:

try-block

finally-block

ex:

----

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(Exception e)

{

e.printStackTrace();

}

finally

{

System.out.println("finally-block");

}

}

}

o/p:

java.lang.ArithmeticException: / by zero

at Test.main(Test.java:7)

finally-block

try and finally combination is valid in java.

ex:

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

finally

{

System.out.println("finally-block");

}

}

}

Q) What is the difference between final, finally and finalize method?

final

-------

It is a modifier which is applicable for variables, methods and classes.

If we declare any variable as final then reassignment of that variable is not possible.

If we declare any method as final then overriding of that method is not possible.

If we declare any class as final then creating child class is not possible.

finally

--------

It is a block which contains cleanup code and it will execute irrespective of exception raised or not.

finalize

----------

It is a method called by garbage collector just before destroying an object for cleanup activity.

throw statement

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

Sometimes we will create exception objects explicitly and handover to JVM manually by using throw statement.

syntax:

-------

throw new ArithmeticException("Don't divide by zero");

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println(10/0);

}

}

Here exception object is created and handover to JVM by main method.

ex:

---

class Test

{

public static void main(String[] args)

{

throw new ArithmeticException("Don't divide by zerooooo");

}

}

Here exception object is created explicitly and handover to JVM manually by using throw statement.

throws statement

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

If any checked exception raised in our program we must and should handle that exception by using try and catch block or by using throws statement.

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

Thread.sleep(3000);

System.out.println("Welcome to Java");

}

catch (InterruptedExceptionie)

{

ie.printStackTrace();

}

}

}

ex:

---

class Test

{

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

{

Thread.sleep(5000);

System.out.println("Welcome to Java World");

}

}

2) Userdefined exceptions

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

Exceptions which are created by the user based on the application requirement are called custom exceptions.

ex:

NoInterestInJavaException

NeedEnjoymentException

ACNotWorkingException

EligibleToVoteException

NotEligibleToVoteException

and etc.

ex

---

import java.util.Scanner;

class EligibleToVoteException extends RuntimeException

{

EligibleToVoteException(String msg)

{

super(msg);

}

}

class NotEligibleToVoteException extends RuntimeException

{

NotEligibleToVoteException(String msg)

{

super(msg);

}

}

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the age :");

int age=sc.nextInt();

if(age<18)

throw new NotEligibleToVoteException("Not Eligible to vote");

else

throw new EligibleToVoteException("Eligible to vote");

}

}

Class 43

java.io package

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

File

=======

File f=new File("abc.txt");

File will check is there any abc.txt file already created or not.

If it is available it simply refers to that file.If it is not created then

it won't create any new file.

ex:

---

import java.io.\*;

class Test

{

public static void main(String[] args)

{

File f=new File("abc.txt");

System.out.println(f.exists());//false

}

}

A File object can be used to create a physical file.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

File f=new File("abc.txt");

System.out.println(f.exists());//false

f.createNewFile();

System.out.println(f.exists());//true

}

}

A File object can be used to create a directory also.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

File f=new File("bhaskar123");

System.out.println(f.exists());//false

f.mkdir();

System.out.println(f.exists());//true

}

}

Q)Write a java program to Create a "cricket123" folder and inside that folder create "abc.txt" file?

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

File f1=new File("cricket123");

f1.mkdir();

File f2=new File("cricket123","abc.txt");

f2.createNewFile();

System.out.println("Please check the location");

}

}

FileWriter

==========

FileWriter is used to write character oriented data into a file.

constructor

--------------

FileWriter fw=new FileWriter(String s);

FileWriter fw=new FileWriter(File f);

ex:

FileWriter fw=new FileWriter("aaa.txt");

or

File f=new File("aaa.txt");

FileWriter fw=new FileWriter(f);

If file does not exist then FileWriter will create a physical file.

Methods

-----------

1)write(int ch)

-----------------

It will insert single character into a file.

2)write(char[] ch)

-----------------

It will insert array of characters into a file.

3)write(String s)

-------------------

It will insert String into a file.

4)flush()

----------

It gives guaranttee that last character of a file is also inserted.

5)close()

-----------

It is used to close the FileWriter object.

ex:

-----

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

FileWriter fw=new FileWriter("aaa.txt");

fw.write(98);// b

fw.write("\n");

char[] ch={'a','b','c'};

fw.write(ch);

fw.write("\n");

fw.write("bhaskar\nsolution");

fw.flush();

fw.close();

System.out.println("Please check the location");

}

}

FileReader

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

It is used to read character oriented data from a file.

constructor

--------------

FileReaderfr=new FileReader(String s);

FileReaderfr=new FileReader(File f);

ex:

FileReaderfr=new FileReader("aaa.txt");

or

File f=new File("aaa.txt");

FileReaderfr=new FileReader(f);

Methods

----------

1)read()

--------

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

2)read(char[] ch)

----------------

It will read collection of characters from a file.

3)close()

---------

It is used to close FileReader object.

ex:1

-------

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

FileReaderfr=new FileReader("aaa.txt");

int i=fr.read();

while(i!=-1)

{

System.out.print((char)i);

i=fr.read();

}

fr.close();

}

}

ex:2

----------

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

FileReaderfr=new FileReader("aaa.txt");

char[] carr=new char[255];

//load the data from file to char array

fr.read(carr);

//reading the data from char array

for(char c:carr)

{

System.out.print(c);

}

fr.close();

}

}

Usage of FileWriter and FileReader is not recommanded to use

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

While inserting the data by using FileWriter ,we need to insert line

seperator(\n) which is very headache for the programmer.

While reading the data by using FileReaderobject ,we need to read character

by character which is not convenient to the programmer.

To overcome this limitation Sun micro system introduced BufferedWriter and BufferedReader.

BufferedWriter

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

It is used to insert character oriented data into a file.

constructor

-----------

BufferedWriterbw=new BufferedWriter(Writer w);

BufferedWriterbw=new BufferedWriter(Writer w,intbuffersize);

BufferedWriter object does not communicate with files directly.

It will take the support of some writer objects.

ex:

FileWriter fw=new FileWriter("bbb.txt");

BufferedWriterbw=new BufferedWriter(fw);

or

BufferedWriterbw=new BufferedWriter(new FileWriter("bbb.txt"));

Methods

---------

1)write(int ch)

-----------------

It will insert single character into a file.

2)write(char[] ch)

-----------------

It will insert array of characters into a file.

3)write(String s)

-------------------

It will insert String into a file.

4)flush()

----------

It gives guaranttee that last character of a file is also inserted.

5)close()

-----------

It is used to close the BufferedWriter object.

6)newLine()

----------

It will insert new line into a file.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

BufferedWriterbw=new BufferedWriter(new FileWriter("bbb.txt"));

bw.write(98);//b

bw.newLine();

char[] ch={'a','b','c'};

bw.write(ch);

bw.newLine();

bw.write("bhaskar");

bw.newLine();

bw.flush();

bw.close();

System.out.println("Please check the location");

}

}

BufferedReader

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

It is enhanced reader to read character oriented data from a file.

constructor

------------

BufferedReaderbr=new BufferedReader(Reader r);

BufferedReaderbr=new BufferedReader(Reader r,intbuffersize);

BufferedReader object can't communicate with files directly.IT will take

support of some reader objects.

ex:

FileReaderfr=new FileReader("bbb.txt");

BufferedReaderbr=new BufferedReader(fr);

or

BufferedReaderbr=new BufferedReader(new FileReader("bbb.txt"));

The main advantage of BufferedReader over FileReader is we can read

character line by line instead of character by character.

methods

---------

1)read()

--------

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

2)read(char[] ch)

----------------

It will read collection of characters from a file.

3)close()

---------

It is used to close BufferedReader object.

4)nextLine()

------------

It is used to read next line from the file.If next line is

not available then it will return null.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

BufferedReaderbr=new BufferedReader(new FileReader("bbb.txt"));

String line=br.readLine();

while(line!=null)

{

System.out.println(line);

line=br.readLine();

}

br.close();

}

}

PrintWriter

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

It is enhanced write to write character oriented data into a file.

constructor

-----------

PrintWriter pw=new PrintWriter(String s);

PrintWriter pw=new PrintWriter(File f);

PrintWriter pw=new PrintWriter(Writer w);

PrintWriter can communicate with files directly and it will take the support of some writer objects.

ex:

PrintWriter pw=new PrintWriter("ccc.txt");

or

PrintWriter pw=new PrintWriter(new File("ccc.txt"));

or

PrintWriter pw=new PrintWriter(new FileWriter("ccc.txt"));

The main advantage of PrintWriter over FileWriter and BufferedWriter is we can insert any type of data.

Assume if we want insert primitive values then PrintWriter is best choice.

methods

------------

write(int ch)

write(char[] ch)

write(String s)

flush()

close()

writeln(int i)

writeln(float f)

writeln(double d)

writeln(String s)

writeln(char c)

writeln(boolean b)

write(int i)

write(float f)

write(double d)

write(String s)

write(char c)

write(boolean b)

ex:

------

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

PrintWriter pw=new PrintWriter("ccc.txt");

pw.write(100);// d

pw.println(100);// 100

pw.print('a');

pw.println(true);

pw.println("hi");

pw.println(10.5d);

pw.flush();

pw.close();

System.out.println("Please check the location");

}

}

various ways to provide input values from the keyboard

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

There are various ways to provide input values from keyboard.

1)Command line argument

2)BufferedReader class

3)Console class

4)Scanner class

1)Command line argument

-------------------------

In command line argument we need to pass our inputs at runtime.

ex:

class Test

{

public static void main(String[] args)

{

String name=args[0];

System.out.println("Welcome : "+name);

}

}

o/p:

javac Test.java

java Test Alan

2)BufferedReader class

--------------------------

BufferedReader class present in java.io package.

BufferedReader class will take InputStreamReader object as a parameter

which is embedded with System.in.

ex:

BufferedReaderbr=

new BufferedReader

(new InputStreamReader(System.in));

To read input values from console we need to readLine() method.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the Name :");

String name=br.readLine();

System.out.println("Welcome : "+name);

}

}

3)Console class

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

Console class present in java.io package.

We can create Console class object by using console() method of System class.

ex:

Console c=System.console();

To read inputs from console we need to use readLine() method.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

Console c=System.console();

System.out.println("Enter the Name :");

String name=c.readLine();

System.out.println("Welcome : "+name);

}

}

4)Scanner class

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

Scanner class present java.util package.

We can create Scanner object class as follow.

ex:

Scanner sc=new Scanner(System.in);

We can read inputs from the console by using following methods.

ex:

next()

nextLine()

nextInt()

nextFloat()

nextDouble()

next().charAt(0);

and etc.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the No :");

int no=sc.nextInt();

System.out.println("Enter the Name :");

String name=sc.next();

System.out.println("Enter the Fee :");

double fee=sc.nextDouble();

System.out.println(no+" "+name+" "+fee);

}

}

Class 44

Generics

==========

Arrays are typesafe. It means we can give guarantee that what type of elements are present in array.

If requirement is there to store String values then it is recommanded to use String[] array.

ex:

String[] sarr=new String[5];

sarr[0]="hi";

sarr[1]="hello";

sarr[2]=10;// C.T.E

At the time of retrieving the data from array we don't need to perform typecasting.

ex:

String[] sarr=new String[5];

sarr[0]="hi";

sarr[1]="hello";

-

-

String val=sarr[0];

Collections are not typesafe.We can't give guarantee that what type of elements are present in Collections.

If requirement is there to store String values then it is never recommanded to use ArrayList because we won't get any compile time error or runtime error but sometimes our program get failure.

ex:

ArrayList al=new ArrayList();

al.add("hi");

al.add("hello");

al.add(10);

At the time of retrieving the data compulsary we need to perform typecasting.

ex:

ArrayList al=new ArrayList();

al.add("hi");

al.add("hello");

al.add(10);

-

-

String s=(String)al.get(0);

To overcome this limitations Sun Micro System introduced Generics concept in 1.5v.

The main objective of generics are

1) To make Collections as typesafe.

2) To avoid typecasting problem.

java.util package

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

All the collections methods and all retealed details about the collections present in java .util.package only.

Q) What is the difference between Arrays and Collections?

Arrays Collections

---------- -------------

It is a collection of homogeneous data It is a collection of homogeneous and hetrogeneous

elements. data elements.

Arrays are fixed in size. Collections are growable in nature.

Performance point of view arrays are Memory point of view Collections are recommanded

recommanded to use. to use.

Arrays not implemented based on data structure Collections are implemented based on data structure

concept.Hence we can't expect any readymade concept.Hence we can expect readymade methods.

method.

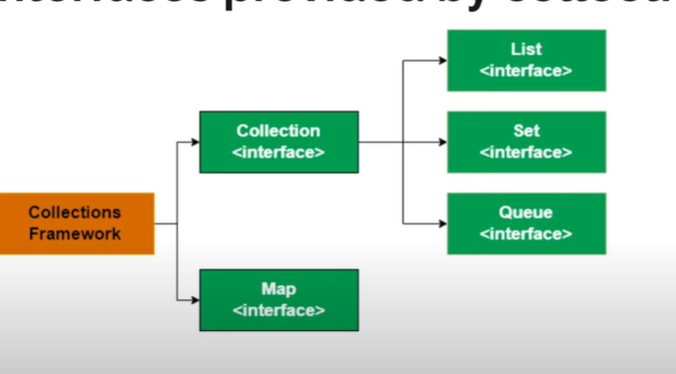
It can hold primitive types and object types. It can hold only object types.

Collection Framework

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

It defines several interfaces and classes to represent group of objects in a single

entity.



Collection

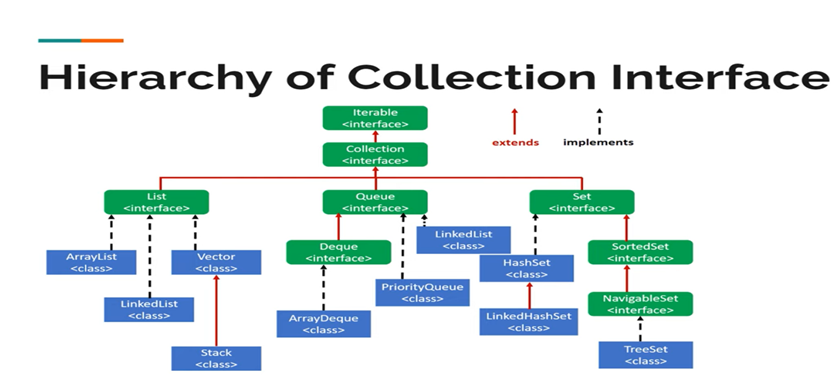
=============

Collection is an interface which is present in java.util package.

It is a root interface for entire collection framework.

If we want to represent group of individual objects in a single entity then we need to use Collection.

Collection interface contains common methods which are applicable for entire Collection objects.

**

ex:

cmd>javapjava.util.Collection

 **add(E e)**: Adds the specified element to the collection.

 **addAll(Collection<? extends E> c)**: Adds all elements from the specified collection to the current collection.

 **clear():** Removes all elements from the collection.

 **contains(Object o):** Checks if the collection contains the specified element.

 **containsAll(Collection<?> c):** Checks if the collection contains all elements from the specified collection.

 **isEmpty():** Checks if the collection is empty.

 **iterator():** Returns an iterator for iterating over the collection.

 **remove(Object o):** Removes the specified element from the collection.

 **removeAll(Collection<?> c):** Removes all elements from the collection that are also in the specified collection.

 **retainAll(Collection<?> c):** Retains only the elements in the collection that are contained in the specified collection.(to find the common elements in the both sets)my own writen

 **size():** Returns the number of elements in the collection.

 **toArray():** Returns an array containing all elements in the collection.

 **toArray(T[] a):** Returns an array containing all elements in the collection; the runtime type of the returned array is that of the specified array.

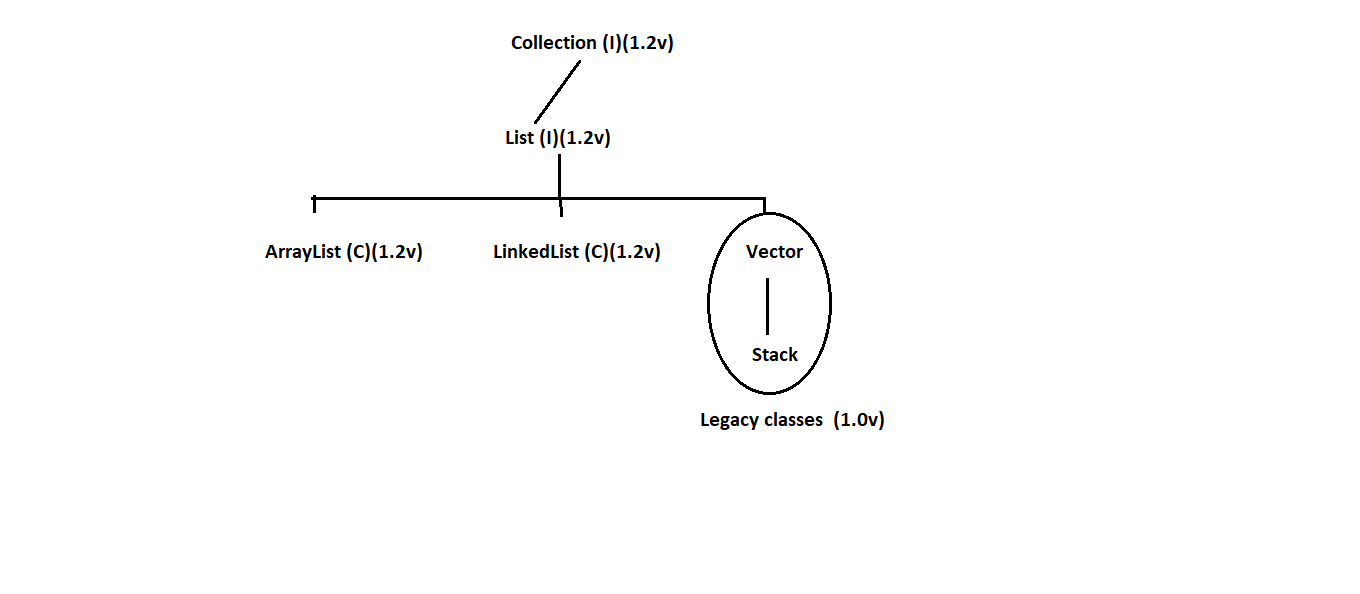
List

======

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are allowed and order is preserved then we need to use List interface.

**Diagram: class44.1**

******

Here are the methods specific to the **List** interface in Java that are not part of the **Collection** interface, along with a brief description of each:

The methods with are present in collection interface we use all methods in all class and interfaces

* **add(int index, E element):** Inserts the specified element at the specified position in the list.
* **addAll(int index, Collection<? extends E> c):** Inserts all elements from the specified collection into the list at the specified position.
* **get(int index):** Returns the element at the specified position in the list.
* **indexOf(Object o):** Returns the index of the first occurrence of the specified element, or -1 if not found.
* **lastIndexOf(Object o):** Returns the index of the last occurrence of the specified element, or -1 if not found.
* **listIterator():** Returns a list iterator for iterating over the list.
* **listIterator(int index):** Returns a list iterator starting at the specified position.
* **remove(int index):** Removes the element at the specified position in the list.
* **set(int index, E element):** Replaces the element at the specified position with the specified element.
* **subList(int fromIndex, int toIndex):** Returns a view of the portion of the list between the specified fromIndex, inclusive, and toIndex, exclusive.

***Arraylist methods***

the methods specific to the ArrayList class in Java that are not part of the Collection interface, along with a brief description of each:

* **ensureCapacity(int minCapacity):** Increases the capacity of the ArrayList to ensure it can hold at least the specified number of elements.
* **trimToSize():** Resizes the internal array to match the current size of the list, reducing memory usage.
* **get(int index):** Returns the element at the specified position in the list.
* **set(int index, E element)**: Replaces the element at the specified position in the list with the specified element.
* **remove(int index):** Removes the element at the specified position in the list.
* **add(int index, E element)**: Inserts the specified element at the specified position in the list.
* **removeRange(int fromIndex, int toIndex):** Removes elements from the list between the specified indices (inclusive).

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("one");

al.add("two");

al.add("three");

System.out.println(al);//[one,two,three]

al.add("one");

System.out.println(al);//[one,two,three,one]

al.add(10);

System.out.println(al); //[one,two,three,one,10]

al.add(null);

System.out.println(al); //[one,two,three,one,10,null]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("one");

al.add("two");

al.add("three");

System.out.println(al);//[one,two,three]

System.out.println(al.isEmpty()); // false

System.out.println(al.contains("one")); // true

al.add(1,"gogo");

System.out.println(al); //[one,gogo,two,three]

al.remove("gogo");

System.out.println(al);//[one,two,three]

al.clear();

System.out.println(al); //[]

}

}

ex

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList<String> al=new ArrayList<String>();

al.add("one");

al.add("two");

al.add("three");

for(int i=0;i<al.size();i++)

{

String s= al.get(i);

System.out.println(s);

}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<String> list=new ArrayList<String>();

list.add("one");

list.add("two");

list.add("three");

for(int i=0;i<list.size();i++)

{

String s= list.get(i);

System.out.println(s);

}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,9,1,2,5);

System.out.println(list);

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList<String> al1=new ArrayList<String>();

al1.add("HTML");

al1.add("CSS");

al1.add("JAVA");

al1.add("ORACLE");

al1.add("FRAMEWORK");

ArrayList<String> al2=new ArrayList<String>();

al2.add("HTML");

al2.add("CSS");

al2.add("DOTNET");

al2.add("SQLSERVER");

al2.add("FRAMEWORK");

al1.retainAll(al2);

System.out.println(al1);//[HTML, CSS, FRAMEWORK]

}

}

LinkedList

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

The underlying data structure is doubly LinkedList.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements List, Serializable,Cloneable and Deque interface.

LinkedList interface contains following methods.

ex:

public E getFirst();

public E getLast();

public E removeFirst();

public E removeLast();

public void addFirst(E);

public void addLast(E);

ex:

the methods specific to the **LinkedList** class in Java that are not part of the **Collection or List** interfaces, along with a brief description of each:

* **addFirst(E e):** Inserts the specified element at the beginning of the list.
* **addLast(E e):** Appends the specified element to the end of the list.
* **getFirst():** Returns the first element of the list.
* **getLast():** Returns the last element of the list.
* **removeFirst():** Removes and returns the first element of the list.
* **removeLast():** Removes and returns the last element of the list.
* **offerFirst(E e):** Adds the specified element at the front of the list (similar to addFirst).
* **offerLast(E e):** Adds the specified element at the end of the list (similar to addLast).
* **pollFirst():** Retrieves and removes the first element of the list, or returns null if the list is empty.
* **pollLast():** Retrieves and removes the last element of the list, or returns null if the list is empty.
* **descendingIterator():** Returns an iterator over the elements in the list in reverse order.

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList ll=new LinkedList();

ll.add("one");

ll.add("two");

ll.add("three");

System.out.println(ll);//[one,two,three]

ll.add("one");

System.out.println(ll);//[one,two,three,one]

ll.add(10);

System.out.println(ll);//[one,two,three,one,10]

ll.add(null);

System.out.println(ll);//[one,two,three,one,10,null]

}

}

ex:

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList ll=new LinkedList();

ll.add("one");

ll.add("two");

ll.add("three");

System.out.println(ll);//[one,two,three]

ll.addFirst("gogo");

ll.addLast("jojo");

System.out.println(ll);//[gogo,one,two,three,jojo]

System.out.println(ll.getFirst());

System.out.println(ll.getLast());

ll.removeFirst();

ll.removeLast();

System.out.println(ll);//[one,two,three]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList<String> ll1=new LinkedList<String>();

ll1.add("one");

ll1.add("two");

ll1.add("three");

System.out.println(ll1);//[one,two,three]

LinkedList<String> ll2=new LinkedList<String>();

ll2.add("raja");

System.out.println(ll2);//[raja]

ll2.addAll(ll1);

System.out.println(ll2);//[raja,one,two,three]

System.out.println(ll2.containsAll(ll1)); // true

ll2.removeAll(ll1);

System.out.println(ll2);//[raja]

}

}

Vector

========

The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements List, Serializable,Cloneable and RandomAccess interface.

All methods present in Vector are synchronized.

Vector class contains following methods.

the methods specific to the **Vector** class in Java that are not part of the **Collection** interface, along with a brief description of each:

* **addElement(E obj):** Adds the specified component (element) to the end of the vector.
* **capacity():** Returns the current capacity of the vector.
* **ensureCapacity(int minCapacity):** Ensures that the vector can hold at least the number of elements specified by minCapacity.
* **elementAt(int index):** Returns the component at the specified index.
* **firstElement():** Returns the first component of the vector.
* **lastElement():** Returns the last component of the vector.
* **removeElement(Object obj):** Removes the first occurrence of the specified element from the vector.
* **removeElementAt(int index):** Removes the component at the specified index.
* **setElementAt(E obj, int index):** Replaces the component at the specified index with the specified element.
* **trimToSize():** Trims the capacity of the vector to be the vector's current size.

ex:

addElement()

removeElementAt()

removeAllElements()

firstElement()

lastElement()

and etc.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector<Integer> v=new Vector<Integer>();

System.out.println(v.capacity()); //10

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(v.firstElement());//1

System.out.println(v.lastElement());//10

v.removeElementAt(5);

System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]

v.removeAllElements();

System.out.println(v); //[]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector<Integer> v=new Vector<Integer>();

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

{

v.add(i);

}

System.out.println(v); //[1,2,3,4,5,6,7,8,9,10]

System.out.println(v.get(0));//1

System.out.println(v.get(v.size()-1));//10

v.remove(5);

System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]

v.clear();

System.out.println(v); //[]

}

}

Class 45

Q) What is the difference between ArrayList and LinkedList?

ArrayList LinkedList

-------------- --------------

The underlying data structure is resizable The underlying data structure is array

or growable array. doubly linkedlist.

ArrayList is better for storing and accessing LinkedList is better for manipulating the data.

the data.

Memory location for ArrayList elements are Memory location for LinkedList elements are not

contigeous. contigeous.

When ArrayList is initialized, a default There is no case of default capacity in LinkedList.

capacity 10 is assigned to it.

Q) What is the difference between ArrayList and Vector?

ArrayList Vector

------------- ------------

No method is synchronized. All methods are synchronized.

At a time only one thread is allowed to operate Multiple threads are allowed to operator

ArrayListobject.Hence it is thread safe. Vector object.Hence it is not thread safe.

Relatively performance is low because waiting time Relatively performance is high because there

of a thread is increasing. is no waiting threads.

It is a non-legacy class. It is a legacy class.

It is introduced in 1.2v. It is introduced in 1.0v.

Q) What is the difference between Collection and Collections?

Collection Collections

---------------- -------------

It is a root interface for entire collection It is a utility class.

framework.

It is used to represent group of individual It defines several utility methods that are operator

objects in a single entity. on Collection.

// Reverse the list Collections.reverse(list);

It contains abstract methods, default methods It contains only static methods.

and static methods.

Q) Write a java program to sort the string?

input:

dog apple boy cat

output:

apple boy cat dog

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="dog apple boy cat";

String[] sarr=str.split(" ");

List<String> list=Arrays.asList(sarr);

Collections.sort(list);

list.forEach(element ->System.out.print(element+" "));

}

}

| **Method Name** | **Description** |
| --- | --- |
| addAll(Collection<? super T> c, T... elements) | Adds all elements to a collection. |
| singleton(E o) | Returns an immutable collection with a single element. |
| emptyList() | Returns an immutable empty list. |
| max(Collection<? extends T> coll) | Finds the maximum element in a collection. |
| min(Collection<? extends T> coll) | Finds the minimum element in a collection. |
| sort(List<T> list) | Sorts a list in natural order or using a comparator. |
| shuffle(List<?> list) | Randomly shuffles elements in a list. |
| reverse(List<?> list) | Reverses the order of elements in a list. |
| copy(List<? super T> dest, List<? extends T> src) | Copies elements from one list to another. |

The methods in Collections are very powerful

Q) How to make ArrayList as synchronized?

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<String> list=new ArrayList<String>();

List<String>synchronizedList = Collections.synchronizedList(list);

synchronizedList.add("one");

synchronizedList.add("two");

synchronizedList.add("three");

synchronizedList.forEach(element ->System.out.print(element+" "));

}

}

Stack

=======

It is a child class of Vector class.

If we depend upon Last In First Out order then we need to use Stack.

constructor

-----------

Stack s=new Stack();

methods

--------

1) push(E)

---------

It is used to push the element to stack.

2) pop()

----------

It is used to pop toppest element from stack.

3) peek()

--------

It will return toppest element of stack.

4) isEmpty()

---------

It will check stack is empty or not.

5) search(Object o)

-----------------

It will return offset value if element is found otherwise it will return -1.

ex:

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Stack<String> s=new Stack<String>();

s.push("A");

s.push("B");

s.push("C");

System.out.println(s);//[A,B,C]

s.pop();

System.out.println(s);//[A,B]

System.out.println(s.peek());// B

System.out.println(s.isEmpty()); // false

System.out.println(s.search("Z")); // -1

System.out.println(s.search("A")); // 2

}

}

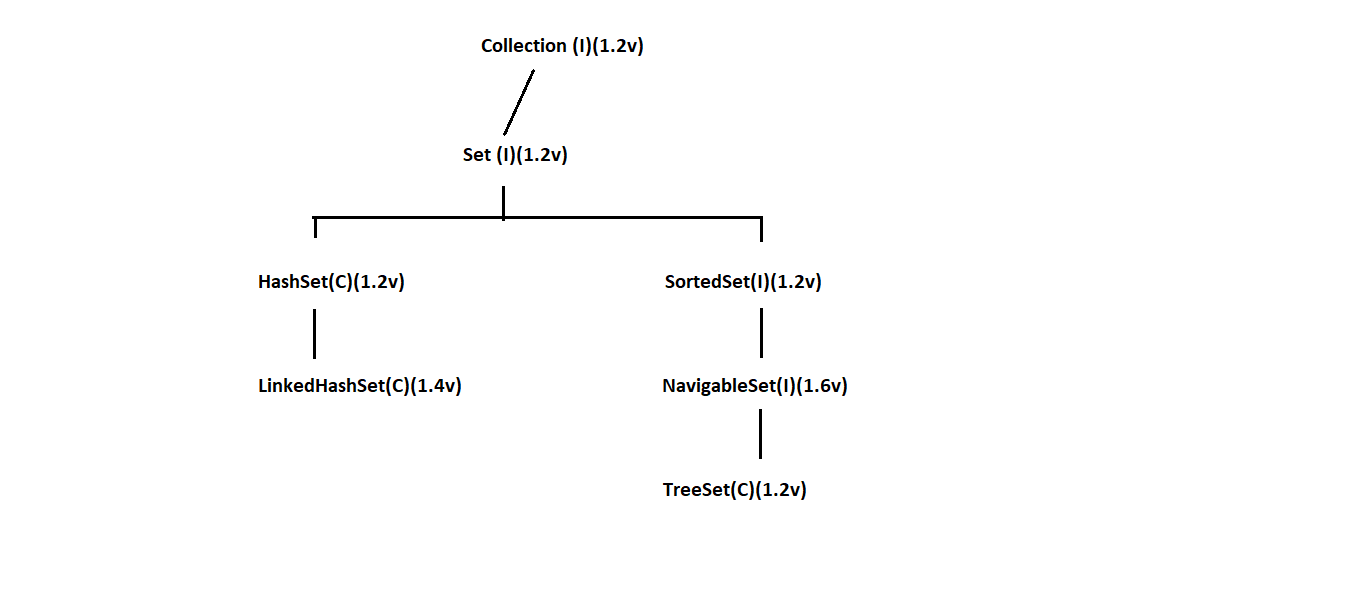
Set

======

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are not allowed and order is not preserved then we need to use Set interface.

**Diagram: class45.1**

******

HashSet

=======

The underlying data structure is Hashtable.

Duplicate objects are not allowed.

Insertion order is not preserved because it will take hash code of an object.

Hetrogeneous objects are allowed.

Null insertion is possible.

Here are all the methods available in the HashSet class in Java, including those inherited from the AbstractSet and Collection classes:

**HashSet Methods**

* **add(E e)**: Adds the specified element to the set if it is not already present.
* **addAll(Collection<? extends E> c):** Adds all elements in the specified collection to the set.
* **clear():** Removes all elements from the set.
* **clone():** Creates a shallow copy of the HashSet instance.
* **contains(Object o)**: Checks if the set contains the specified element.
* **containsAll(Collection<?> c):** Checks if the set contains all elements in the specified collection.
* **isEmpty():** Checks if the set is empty.
* **iterator():** Returns an iterator over the elements in the set.
* **remove(Object o):** Removes the specified element from the set if it is present.
* **removeAll(Collection<?> c):** Removes all elements in the set that are also in the specified collection.
* **retainAll(Collection<?> c):** Retains only the elements in the set that are contained in the specified collection.
* **size():** Returns the number of elements in the set.
* **toArray():** Returns an array containing all elements in the set.
* **toArray(T[] a):** Returns an array containing all elements in the set; the runtime type of the returned array is that of the specified array.

**Additional Information**

* **equals(Object o):** Compares the specified object with this set for equality.
* **hashCode():** Returns the hash code value for the set.
* **spliterator():** Creates a spliterator over the elements in the set.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashSet hs=new HashSet();

hs.add("one");

hs.add("six");

hs.add("nine");

System.out.println(hs);//[nine, six, one]

hs.add("one");

System.out.println(hs);//[nine, six, one]

hs.add(10);

System.out.println(hs);//[nine, six, one, 10]

hs.add(null);

System.out.println(hs);//[null, nine, six, one, 10]

}

}

LinkedHashSet

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

It is a child class of HashSet class.

LinkedHashSet is exactly same as HashSet class with following differences.

Same methods as hashset and linkedset

HashSet LinkedHashSet

------------- ----------------

The underlying data structure is Hashtable. The underlying data structure is Hashtable and

LinkedList.

Insertion order is not preserved. Insertion order is preserved.

It is introduced in 1.2v. It is introduced in 1.4v.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashSetlhs=new LinkedHashSet();

lhs.add("one");

lhs.add("six");

lhs.add("nine");

System.out.println(lhs);//[one, six, nine]

lhs.add("one");

System.out.println(lhs);//[one, six, nine]

lhs.add(10);

System.out.println(lhs);//[one, six, nine, 10]

lhs.add(null);

System.out.println(lhs);//[one, six, nine, 10, null]

}

}

Class 46

Q) Write a java program to display distinct elements from given array?

input:

1 2 2 3 3 3 4 4 4 4

output:

1 2 3 4

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

int[] arr={1,2,2,3,3,3,4,4,4,4};

Set<Integer> set = new LinkedHashSet<Integer>();

//for each loop

for(int i:arr)

{

set.add(i);

}

set.forEach(element ->System.out.print(element+" "));

}

}

TreeSet

=======

The underlying datastructure is Balanced Tree.

Duplicate objects are not allowed.

Insertion order is not preserved because it will take sorting order.

Hetrogeneous objects are not allowed. If we try to insert hetrogeneous objects then we will get ClassCastException.

Null insertion is not possible. If we insert null then we will get NullPointerException.

the methods that are specific to the TreeSet class in Java, which are not found in HashSet:

**TreeSet-Specific Methods**

* **first()**: Returns the first (lowest) element in the set.
* **last()**: Returns the last (highest) element in the set.
* **floor(E e):** Returns the greatest element in the set that is less than or equal to the specified element, or null if there is no such element.
* **ceiling(E e):** Returns the least element in the set that is greater than or equal to the specified element, or null if there is no such element.
* **higher(E e):** Returns the least element in the set strictly greater than the specified element, or null if there is no such element.
* **lower(E e):** Returns the greatest element in the set strictly less than the specified element, or null if there is no such element.
* **comparator():** Returns the comparator used to sort the elements in the set, or null if the set uses the natural ordering of its elements.
* **pollFirst():** Retrieves and removes the first (lowest) element, or returns null if the set is empty.
* **pollLast():** Retrieves and removes the last (highest) element, or returns null if the set is empty.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSetts=new TreeSet();

ts.add(10);

ts.add(1);

ts.add(5);

ts.add(7);

System.out.println(ts); // [1,5,7,10]

ts.add(1);

System.out.println(ts); // [1,5,7,10]

//ts.add("one");

//System.out.println(ts); // R.E ClassCastException

//ts.add(null);

//System.out.println(ts); // R.E NullPointerException

}

}

Q) What is the difference between Comparable and Comparator interface?

Comparable

----------

Comparable is an interface which is present in java.lang package.

Comparable interface contains only one method i.ecompareTo() method.

If we depend upon default natural sorting order then we need to use Comparable interface.

ex:

obj1.compareTo(obj2)

It returns -ve if obj1 comes before obj2

It returns +ve if obj1 comes after obj2

it returns 0 if both objects are same

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("A".compareTo("Z")); // -25

System.out.println("Z".compareTo("A")); // 25

System.out.println("K".compareTo("K")); // 0

}

}

Comparator

-----------

Comparator is an interface which is present in java.util package.

Comparator interface contains following two methods i.ecompare() and equals() method.

If we depend upon customized sorting order then we need to use Comparator interface.

ex:

public int compare(Object obj1,Object obj2)

It returns +ve if obj1 comes before obj2

It returns -ve if obj1 comes after obj2

it returns 0 if both objects are same

Implementation of equals() method is optional because it is present in Object class which is available to the class through inheritance.

Implementation of compare() method is mandatory.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet<Integer>ts=new TreeSet<Integer>(new MyComparator());

ts.add(10);

ts.add(1);

ts.add(5);

ts.add(7);

System.out.println(ts);//[10, 7, 5, 1]

}

}

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;

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet<Integer>ts=new TreeSet<Integer>(new MyComparator());

ts.add(10);

ts.add(1);

ts.add(5);

ts.add(7);

System.out.println(ts);//[1, 5, 7, 10]

}

}

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;

}

}

Q) Write a java program to compare two dates?

import java.time.\*;

class Test

{

public static void main(String[] args)

{

LocalDate date1=LocalDate.now();

LocalDate date2=LocalDate.of(2024,10,15);

if(date1.compareTo(date2)>0)

System.out.println("date1 is biggest");

else if(date1.compareTo(date2)<0)

System.out.println("date2 is biggest");

else

System.out.println("Both are same");

}

}

Map

====

It is not a child interface of Collection interface.

If we want to represent group of individual objects in key and value pair then we need to use

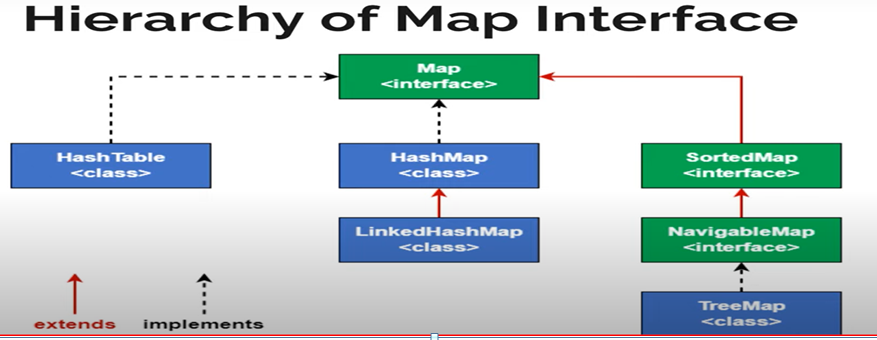
Map interface.

Key and value both must be objects.

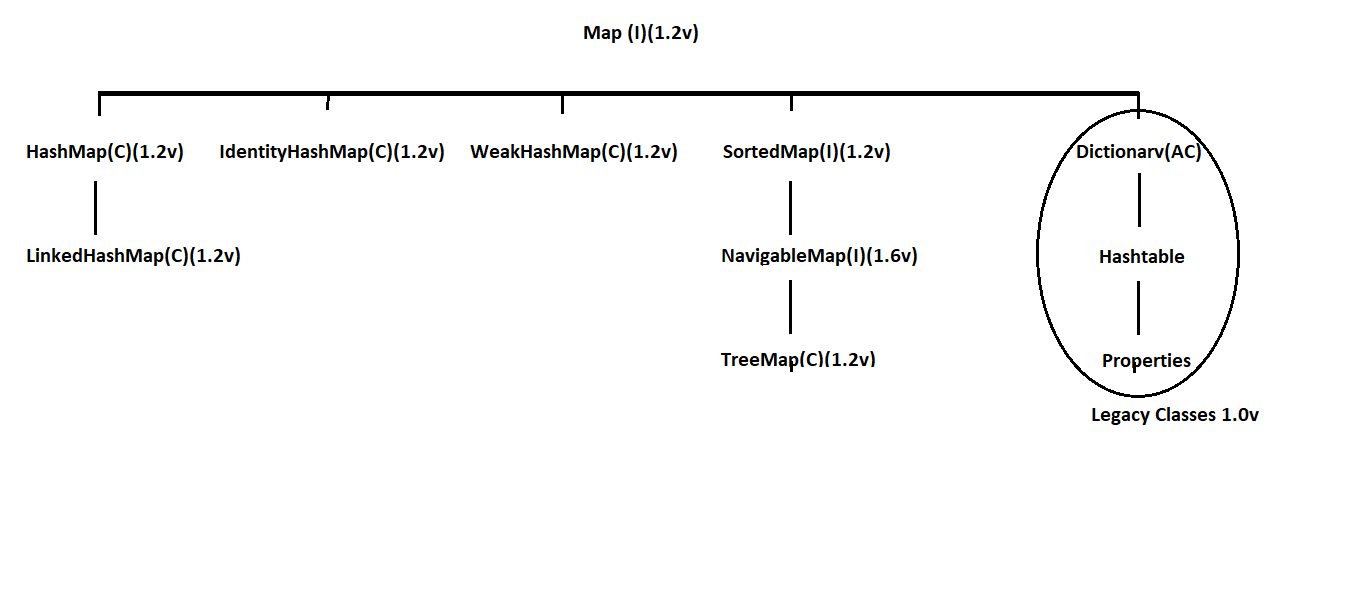
Key can't be duplicate but value can be duplicate.

Each key and value pair is called single/one entry.

* **Map Methods**
* put(K key, V value): Associates the specified value with the specified key in the map.
* putAll(Map<? extends K, ? extends V> m): Copies all of the mappings from the specified map to this map.
* **get(Object key):** Returns the value to which the specified key is mapped, or null if the map contains no mapping for the key.
* **remove(Object key):** Removes the mapping for the specified key from the map if present.
* **containsKey(Object key)**: Checks if the map contains a mapping for the specified key.
* **containsValue(Object value):** Checks if the map maps one or more keys to the specified value.
* **keySet():** Returns a set view of the keys contained in the map.
* **values():** Returns a collection view of the values contained in the map.
* **entrySet():** Returns a set view of the mappings contained in the map.
* **size():** Returns the number of key-value mappings in the map.
* **isEmpty():** Checks if the map is empty.
* **clear():** Removes all mappings from the map.
* **putIfAbsent(K key, V value):** Associates the specified value with the specified key if the key is not already associated with a value.
* **remove(Object key, Object value):** Removes the entry for the specified key only if it is currently mapped to the specified value.
* **replace(K key, V value): Replaces** the entry for the specified key only if it is currently mapped to some value.
* **replace(K key, V oldValue, V newValue):** Replaces the entry for the specified key only if currently mapped to the specified old value.
* **forEach(BiConsumer<? super K, ? super V> action):** Performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
* **computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction):** If the specified key is not already associated with a value, compute its value using the given mapping function.
* **computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction):** If the specified key is already associated with a value, compute a new mapping based on the existing value.
* **compute(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)**: Computes a value for the specified key using the given remapping function.
* **merge(K key, V value, BiFunction<? super V, ? super V, ? extends V> remappingFunction)**: If the specified key is not already associated with a value, associates it with the given value. Otherwise, replaces the value with the results of the given remapping function.

****

**Diagram: class46.1**

******

HashMap

=========

The underlying data structure is Hashtable.

Duplicate key is not allowed but value can be duplicate.

Insertion order is not preserved because it will take hash code of a key.

Hetrogeneous objects are allowed for both key and value.

Null insertion is possible for both key and value.

**HashMap-Specific Methods**

1. **putIfAbsent(K key, V value)**: Adds the specified key-value pair if the key is not already associated with a value.
2. **remove(Object key, Object value):** Removes the entry for the specified key only if it is currently mapped to the specified value.
3. **replace(K key, V value):** Replaces the value for the specified key if it is currently mapped to a value.
4. **replace(K key, V oldValue, V newValue):** Replaces the value for the specified key only if it is currently mapped to the specified old value.
5. **computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction):** Computes a value for the specified key if it is not already associated with a value.
6. **computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction):** Computes a new mapping for the specified key if it is already associated with a value.
7. **compute(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction):** Computes a new value for the specified key using the given remapping function.
8. **merge(K key, V value, BiFunction<? super V, ? super V, ? extends V> remappingFunction):** Merges the specified value with the existing value for the specified key using the provided function.
9. /\*  . Sort a Map by Keys
10. Problem:Given a map of strings to integers, sort the map by its keys.
11. Input:map = {banana=1, apple=5, cherry=3, date=4}
12. Output:{apple=5, banana=1, cherry=3, date=4} \*/
13. public static void sub6()
14. {
15. HashMap<String,Integer> hm=new HashMap<>(Map.of("banana",1, "apple",5, "cherry",3, "date",4));
16. List <Map.Entry<String,Integer>>li=new ArrayList<>(hm.entrySet());
17. li.sort(Map.Entry.comparingByKey());
18. System.out.println(li);
19. }
20. //another method
21. public static void sub6()
22. {
23. TreeMap<String,Integer> hm=new TreeMap<>(Map.of("banana",1, "apple",5, "cherry",3, "date",4));
24. System.out.println(hm);
25. }

ex:

--

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashMap hm=new HashMap();

hm.put("one","raja");

hm.put("six","jose");

hm.put("nine","brook");

hm.put("five","alan");

System.out.println(hm);//{nine=brook, six=jose, one=raja, five=alan}

hm.put("one","gogo");

System.out.println(hm);//{nine=brook, six=jose, one=gogo, five=alan}

hm.put(10,100);

System.out.println(hm);//{nine=brook, six=jose, one=gogo, 10=100, five=alan}

hm.put(null,null);

System.out.println(hm);//{null=null, nine=brook, six=jose, one=gogo, 10=100, five=alan}

}

}

LinkedHashMap

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

LinkedHashMap is a child class of HashMap class.

LinkedHashMap is exactly same as HashMap class with following differences.

**LinkedHashMap**

1. **Order of Iteration**: Unlike HashMap, LinkedHashMap maintains a predictable iteration order (insertion order).
2. **Access Order:** You can create a LinkedHashMap that maintains access order by specifying true as the second argument in its constructor, allowing it to be used as a cache.

* **accessOrder:** If true, the order in which elements are iterated is the order in which they were last accessed, rather than the order in which they were inserted.

HashMap LinkedHashMap

------------ ---------------

The underlying data structure is Hashtable. The underlying data structure is Hashtable and

LinkedList.

Insertion order is not preserved. Insertion order is preserved.

It is introduced in 1.2v. It is introduced in 1.4v.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashMaplhm=new LinkedHashMap();

lhm.put("one","raja");

lhm.put("six","jose");

lhm.put("nine","brook");

lhm.put("five","alan");

System.out.println(lhm);//{one=raja, six=jose, nine=brook, five="alan"}

lhm.put("one","gogo");

System.out.println(lhm);//{one=gogo, six=jose, nine=brook, five="alan"}

lhm.put(10,100);

System.out.println(lhm);//{one=gogo, six=jose, nine=brook, five="alan", 10=100}

lhm.put(null,null);

System.out.println(lhm);//{one=gogo, six=jose, nine=brook, five="alan", 10=100, null=null}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Map<String,String> map=new LinkedHashMap<String,String>();

map.put("one","raja");

map.put("two","ravi");

map.put("three","ramana");

Set s=map.keySet();

System.out.println(s);

Collection c=map.values();

System.out.println(c);

Set s1=map.entrySet();

System.out.println(s1);

}

}

**Class 47**

**TreeMap**

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

The underlying data structure is RED BLACK TREE.

Duplicate key is not allowed but value can be duplicate.

Insertion order is not preseved because it takes sorting order of key.

If we depend upon default natural sorting order then key must be homogeneous and Comparable.

If we depend upon customized sorting order then key must be hetrogeneous and Non-Comparable.

Key can't be null but value can be null.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeMap<Integer,String> tm=new TreeMap<Integer,String>();

tm.put(10,"ten");

tm.put(1,"one");

tm.put(5,"five");

tm.put(7,"seven");

System.out.println(tm); // {1=one, 5=five, 7=seven, 10=ten}

tm.put(1,"gogo");

System.out.println(tm);//{1=gogo, 5=five, 7=seven, 10=ten}

tm.put(4,null);

System.out.println(tm); //{1=gogo, 4=null, 5=five, 7=seven, 10=ten}

tm.put(null,"four");

System.out.println(tm); //R.ENullPointerException

}

}

Hashtable

=========

The underlying data structure is Hashtable.

Duplicate key is not allowed but value can be duplicate.

Insertion order is not preserved because it takes descending order of key.

Hetrogeneous objects are allowed for both key and value.

Null insertion is not possible for both key and value.

ex:

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Hashtableht=new Hashtable();

ht.put(1,"one");

ht.put(10,"ten");

ht.put(5,"five");

ht.put(3,"three");

System.out.println(ht);//{10=ten, 5=five, 3=three, 1=one}

ht.put(1,"gogo");

System.out.println(ht);//{10=ten, 5=five, 3=three, 1=gogo}

ht.put("four",4);

System.out.println(ht);//{10=ten, 5=five, four=4, 3=three, 1=gogo}

//ht.put(6,null);

//System.out.println(ht); //R.ENullPointerException

//ht.put(null,"six");

//System.out.println(ht); // R.E NullPointerException

}

}

**Diagram: class47.1**

***A white grid with black text

Description automatically generated***

Q) Write a java program to display number of occurance of a given string?

input:

This is is java class class

output:

This=1 is=2 java=1 class=2

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="This is is java class class";

String[] sarr=str.split(" ");

Map<String,Integer> map=new LinkedHashMap<String,Integer>();

for(String s:sarr)

{

if(map.get(s)!=null)

{

map.put(s,map.get(s)+1);

}

else

{

map.put(s,1);

}

}

map.forEach((key,value)->System.out.print(key+"="+value+" "));

}

}

Q) Write a java program to display number of occurance of a given string?

input:

java

output:

j=1 a=2 v=1

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="java";

char[] carr=str.toCharArray();

Map<Character,Integer> map=new LinkedHashMap<Character,Integer>();

for(char c:carr)

{

if(map.get(c)!=null)

{

map.put(c,map.get(c)+1);

}

else

{

map.put(c,1);

}

}

map.forEach((key,value)->System.out.print(key+"="+value+" "));

}

}

Q) Write a java program to display lucky number from given array?

input:

1 1 2 3 3 3 4 4

output:

3

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

int[] arr={1,1,2,3,3,3,4,4};

System.out.println(luckyNumber(arr));

}

//callie method

public static int luckyNumber(int[] arr)

{

Map<Integer,Integer> map=new HashMap<Integer,Integer>();

for(int i:arr)

{

if(map.containsKey(i))

{

map.put(i,map.get(i)+1);

}

else

{

map.put(i,1);

}

}

int x=0;

int max=-1;

for(Map.Entry<Integer,Integer>entry:map.entrySet())

{

if(entry.getKey() == entry.getValue())

{

x = entry.getKey();

max = Math.max(x,max);

}

}

return max;

}

}

Types of Cursors in java

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

Cursors are used to read the objects one by one from Collections.

We have three types of cursors.

1) Enumeration

2) Iterator

3) ListIterator

1) Enumeration

--------------

It is used to read the objects one by one from legacy Collection objects.

We can create Enumeration object as follow.

ex:

Enumeration e=v.elements();

Enumeration interface contains two methods.

ex:

public booleanhasMoreElements();

public Object nextElement();

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector v=new Vector();

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

{

v.add(i);

}

System.out.println(v); //[1,2,3,4,5,6,7,8,9,10]

Enumeration e=v.elements();

while(e.hasMoreElements())

{

Integer i=(Integer)e.nextElement();

System.out.print(i+" ");

}

}

}

Limitations with Enumeration

---------------------------

Enumeration is used to read objects one by one from legacy Collections objects only.Hence it is a not a universal cursor.

Using Enumeration we can perform read operation but not remove operation.

To overcome this limitation Sun Micro System introduced Iterator.

2) Iterator

------------

Iterator is used to read objects one by one from any Collection object.Hence it is a universal cursor.

Using Iterator we can perform read and remove operations.

We can create Iterator object as follow.

ex:

Iterator itr=al.iterator();

Iterator interface contains following three methods.

ex:

public booleanhasNext();

public Object next();

public void remove();

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

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

{

al.add(i);

}

System.out.println(al);//[1,2,3,4,5,6,7,8,9,10]

Iterator itr=al.iterator();

while(itr.hasNext())

{

Integer i=(Integer)itr.next();

if(i%2==0)

System.out.println(i+" ");

else

itr.remove();

}

System.out.println(al);//[2,4,6,8,10]

}

}

Limitations with Iterator

------------------------

Using Enumeration and Iterator we can read objects only in forward direction but not in backward direction.Hence they are not bi-directional cursors.

Using Iterator we can perform read and remove operations but not adding and replacement of new object.

To overcome this limitation Sun Micro System introduced ListIterator.

3) ListIterator

-------------

It is a child interface of Iterator interface.

ListIterator is used to read objects only from List Collection objects.

Using Listiterator we can perform read , remove, adding and replacement of new objects.

We can create ListIterator object as follow.

ex:

ListIteratorlitr=al.listIterator();

ListIterator interface contains following methods.

ex:

public booleanhasNext()

public Object next()

public void remove()

public booleanhasPrevious()

public Object previous()

public int nextIndex()

public int previousIndex()

public void set(Object o)

public void add(Object o)

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIteratorlitr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

System.out.println(s);

}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIteratorlitr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

if(s.equals("bala"))

{

litr.remove();

}

}

System.out.println(al); //[venki,chiru,nag]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIteratorlitr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

if(s.equals("bala"))

{

litr.add("charan");

}

}

System.out.println(al); //[bala, charan, venki, chiru, nag]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIteratorlitr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

if(s.equals("bala"))

{

litr.set("charan");

}

}

System.out.println(al); //[charan, venki, chiru, nag]

}

}

**Diagram: class47.2**

***A diagram of data structure

Description automatically generated***

**Class 48**

Multithreading

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

Q) What is the difference between Thread and Process?

Thread

------

It is a leight weight sub process.

We can run multiple threads concurently.

One thread can communicate with another thread.

Process

-------

It is a collection of threads.

We can run multiple process concurently.

One process can't communicate with another process.

Multitasking

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

Executing several task simultenously such concept is called multitasking.

We have two types of multitasking.

1) Thread based multitasking

2) Process based multitasking

1) Thread based multitasking

------------------------------

Executing several task simultenously where each task is a same part of a program.

It is best suitable for programmatic level.

2) Process based multitasking

---------------------------

Executiong several task simultenously where each task is a independent process.

It is best suitable for OS level.

MultiThreading

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

Executing several threads simultenously such concept is called multithreading.

In multithreading only 10% of work should be done by a programmer and 90% of work will be done by JAVA API.

The main important application area of multithreading are.

1) To implements multi media graphics.

2) To develop video games.

3) To develop animations.

Ways to start a thread in java

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

There are two ways to start a thread in java.

1) By extending Thread class

2) By implementing Runnable interface

1) By extending Thread class

----------------------------

class MyThread extends Thread

{

public void run()

{

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

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

case1: Thread Schedular

-----------------------

If multiple threads are waiting for execution which thread has to be executed will decided by thread schedular.

What algorithm, behaviour, mechanism used by thread schedular is depends upon JVM vendor.

Hence we can't expect any execution order or exact output in multithreading.

case2: Difference between t.start() method and t.run() method

--------------------------------------------------------------

If we invoke t.start() method then a new thread will be created which is responsible to execute run() method automatically.

class MyThread extends Thread

{

public void run()

{

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

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

If we invoke t.run() method then no new thread will be created but run() method will execute just like normal method.

ex:

class MyThread extends Thread

{

public void run()

{

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

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//no new thread

t.run();

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

{

System.out.println("Parent-Thread");

}

}

}

case3: If we won't override run() method

-----------------------------------

If we won't override run() method then Thread class run() method will execute automatically.

Thread class run() method is empty implementation.Hence we won't get any output from child thread.

ex:

---

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

case4: If we overload run() method

----------------------------------

If we overload run() method then Thread class start() method always execute run() method with zero arguments only.

ex:

---

class MyThread extends Thread

{

public void run(int i)

{

System.out.println("int-arg method");

}

public void run()

{

System.out.println("0-arg method");

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

case 5: Life cycle of a thread

-------------------------------

**Diagram: class48.1**

***A diagram of a network

Description automatically generated***

Once if we create thread then our thread will be in new or born state.

Once if we call t.start() method our thread goes to ready/runnable state.

If thread schedular allocates to CPU then our thread enters to running state.

Once the run() method execution is completed then our thread goes to dead state.

2) By implementing Runnable interface

---------------------------------------

class MyRunnable implements Runnable

{

public void run()

{

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

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

MyRunnable r=new MyRunnable();

Thread t=new Thread(r); // r is a targatable interface

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

Setting and getting name of a thread

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

In java, every thread has a name. Explicitly provided by the programmer or automatically generated by the JVM.

We have following methods to set and get name of a thread.

ex:

public final void setName(String name);

public final String getName();

ex:

---

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

System.out.println(Thread.currentThread().getName());// main

MyThread t=new MyThread();

System.out.println(t.getName());//Thread-0

Thread.currentThread().setName("Parent-Thread");

System.out.println(Thread.currentThread().getName());//Parent-Thread

t.setName("Child-Thread");

System.out.println(t.getName());//Child-Thread

}

}

Thread priority

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

In java, every thread has a priority explicitly provided by the programmer and automatically generated by JVM.

The valid range of thread priority is 1 to 10. Where 1 is a least priority and 10 is a highest priority.

If we take more then 10 priority then we will get IllegalArgumentException.

Thread class defines following standard constants as thread priority.

ex:

Thread.MAX\_PRIORITY - 10

Thread.NORM\_PRIORITY - 5

Thread.MIN\_PRIORITY - 1

We don't have such constants like LOW\_PRIORITY and HIGH\_PRIORITY.

A thread which is having highest priority will be executed first.

If multiple threads having same priority then we can't expect any execution order.

Thread schedular uses thread priorities while allocating to CPU.

We have following methods to set and get thread priority.

ex:

public final void setPriority(int priority)

public final int getPriority()

ex

---

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

System.out.println(Thread.currentThread().getPriority());// 5

MyThread t=new MyThread();

System.out.println(t.getPriority());//5

Thread.currentThread().setPriority(10);

System.out.println(Thread.currentThread().getPriority());//10

t.setPriority(4);

System.out.println(t.getPriority());//4

//t.setPriority(11);//R.E IllegalArgumentException

}

}

Daemon Thread

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

Daemon thread is a service provider thread which provides services to user threads.

Life of daemon thread is depends upon user threads because when user threads died then daemon thread will die automatically.

There are many daemon thread are running internally like Garbage collector , finalizer and etc.

We can start a daemon thread by using setDeamon(true) method.

To check a thread is a daemon or not we will use isDaemon() method.

ex:

---

class MyThread extends Thread

{

public void run()

{

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

{

System.out.println(Thread.currentThread().isDaemon());

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread t=new MyThread();

t.setDaemon(true);

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

Assignment

===========

class Employee

{

private int empId;

private String empName;

private double empSal;

//parameterized constructor

//setter and getter methods

}

interface IEmployee

{

public List<Employee>getAllEmployees();

}

class EmployeeImpl implements IEmployee

{

public List<Employee>getAllEmployees()

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(101,'raja',1000d));

list.add(new Employee(102,'ravi',2000d));

list.add(new Employee(103,'ramana',3000d));

return list;

}

}

**Class 49**

**Various ways to prevent a thread from execution**

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

There are three ways to prevent(stop) a thread from execution.

1) yield()

2) join()

3) sleep()

**1) yield()**

**-----------**

It pause the current execution thread and gives the chance to other threads having same

priority.

If there is no waiting threads or low priority threads then same thread will continue

it's execution.

If multiple waiting threads having same priority then we can't expect any execution order.

**ex:**

public static native void yield();

**Diagram: class49.1**

***A diagram of a running process

Description automatically generated***

class MyThread extends Thread

{

public void run()

{

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

{

Thread.currentThread().yield();

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread t=new MyThread();

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

**2) join()**

**----------**

If a thread wants to wait untill the completion of some other threads then we need to

join().

A join() method throws one checked exception called InterruptedException so we must

and should handle that exception by using try and catch block or by using throws stmt.

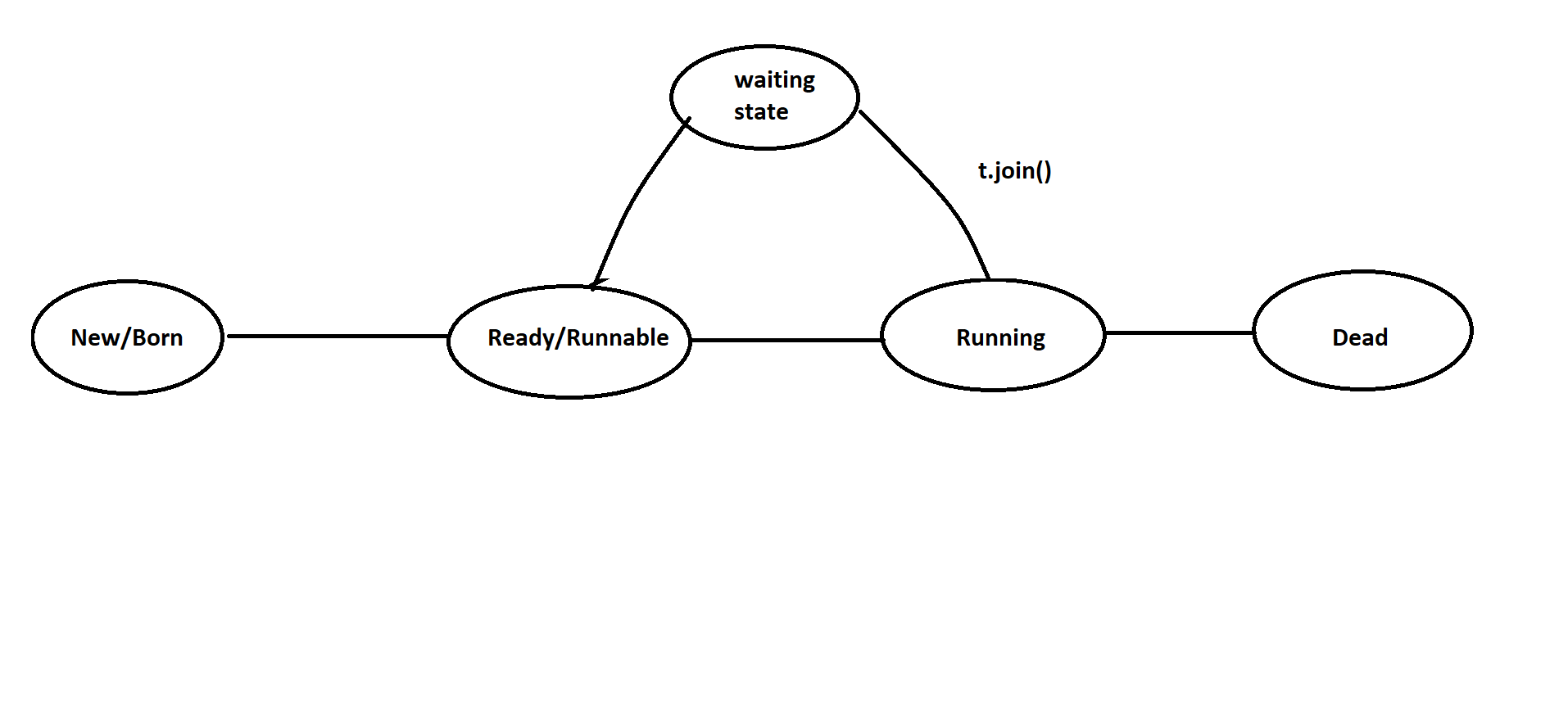
ex:

public final void join()throws InterruptedException

public final void join(long ms)throws InterruptedException

public final void join(long ms,int ns)throws InterruptedException

**Diagram: class49.2**

******

class MyThread extends Thread

{

public void run()

{

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

{

System.out.println("Child-Thread");

}

}

}

class Test

{

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

{

MyThread t=new MyThread();

t.start();

t.join();

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

{

System.out.println("Parent-Thread");

}

}

}

**3) sleep()**

**-----------**

If a thread don't want to perform any operation on perticular amount of time then we

need to use sleep() method.

A sleep() method throws one checked exception so we must and should handle that exception

by using try and catch block or by using throws statement.

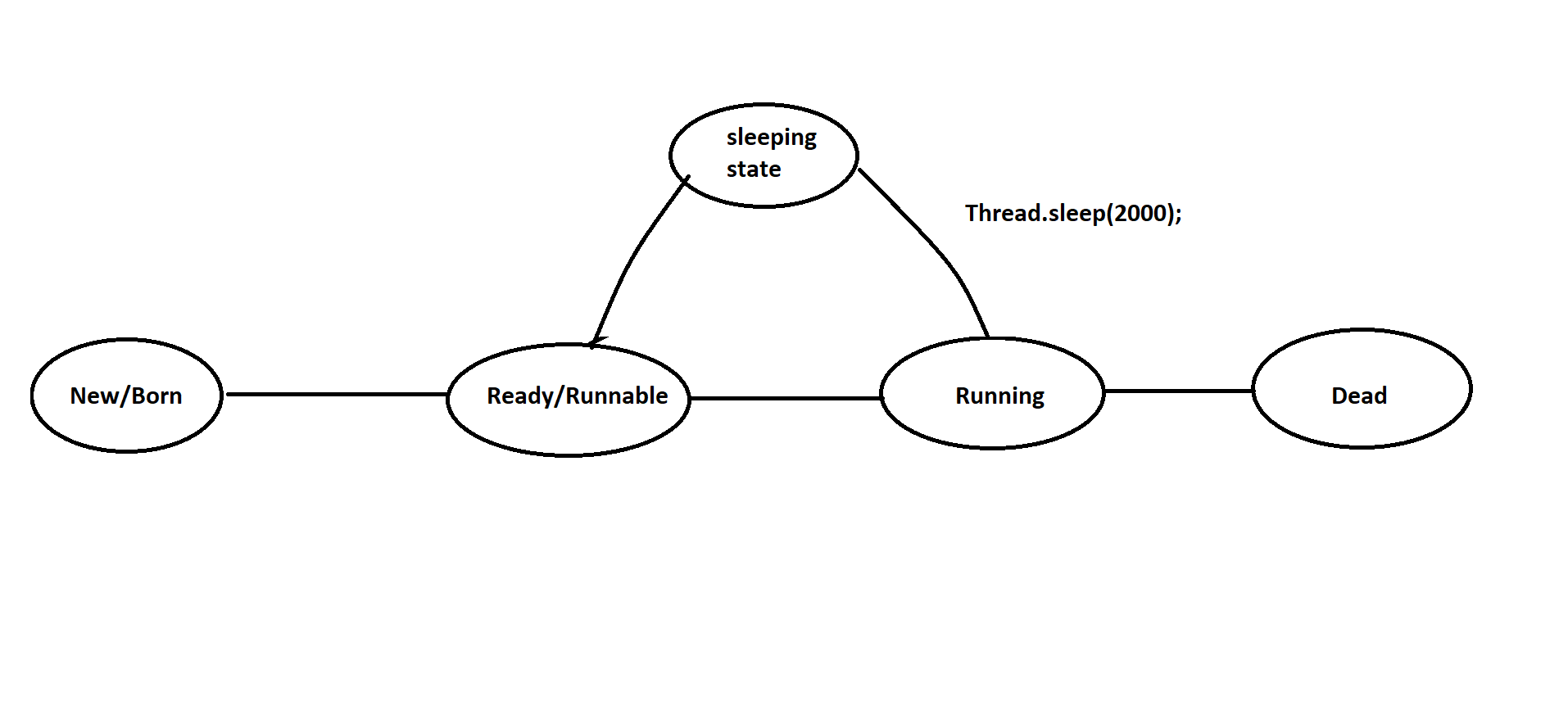
ex:

public static native void sleep()throws InterruptedException

public static native void sleep(long ms)throws InterruptedException

public static native void sleep(long ms,int ns)throws InterruptedException

**Diagram: class49.3**

**

ex:

---

class MyThread extends Thread

{

public void run()

{

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

{

System.out.println("Child-Thread");

try

{

Thread.sleep(2000);

}

catch (InterruptedExceptionie)

{

ie.printStackTrace();

}

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread t=new MyThread();

t.start();

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

{

System.out.println("Parent-Thread");

}

}

}

Problem without synchronization

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

If there is no sychronization then we will face following problems.

1) Data inconsistency

2) Thread interference

ex:

---

class Table

{

void printTable(int n)

{

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

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedExceptionie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

synchronization

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

A synchronized keyword is applicable for methods and blocks.

A synchronization is allowed one thread to execute given object.Hence we achieve thread safety.

The main advantage of synchronization is we solve data inconsistence problem.

The main disadvantage of synchronization is ,it will increase waiting time of a thread which reduce the performance of the system.

If there is no specific requirement then it is never recommanded to use synchronization concept.

synchronization internally uses lock mechanism.

Whenever a thread wants to access object , first it has to acquire lock of an object and thread will release the lock when it completes it's task.

When a thread wants to execute synchronized method.It automatically gets the lock of an object.

When one thread is executing synchronized method then other threads are not allowed to execute other synchronized methods in a same object concurently.But other threads are allowed to execute non-synchronized method concurently.

ex:

class Table

{

synchronized void printTable(int n)

{

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

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedExceptionie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

synchronized block

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

If we want to perform synchronization on specific resource of a program then we need to use

synchronization.

ex:

If we have 100 lines of code and if we want to perform synchronization only for

10 lines then we need to use synchronized block.

If we keep all the logic in synchronized block then it will act as a synchronized method.

**ex:**

class Table

{

void printTable(int n)

{

synchronized(this)

{

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

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedExceptionie)

{

ie.printStackTrace();

}

}

}//sync

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

3)Static synchronization

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

In static synchronization the lock will be on class but not on object.

If we declare any static method as synchronized then it is called static synchronization method.

ex:

class Table

{

static synchronized void printTable(int n)

{

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

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedExceptionie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

public void run()

{

Table.printTable(5);

}

}

class MyThread2 extends Thread

{

public void run()

{

Table.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

MyThread1 t1=new MyThread1();

MyThread2 t2=new MyThread2();

t1.start();

t2.start();

}

}

Inter-Thread Communication

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

Two threads can communicate with one another by using wait(),notify() and notifyAll() method.

The Thread which is expecting updations it has to wait() method and the thread which is performing updations it has to call notify() method.

wait(),notify() and notifyAll() method present in Object class but not in Thread class.

To call wait(),notify() and notifyAll() method our current thread must be in a synchronized area otherwise we will get IllegalMonitorStateException.

Once a thread calls wait() method on a given object ,1st it will release the lock of that object immediately and entered into waiting state.

Once a thread calls notify() and notifyAll() method on a given object.It will release the lock of that object but not immediately.

Except wait(),notify() and notifyAll() method ,there is no such concept where lock release can happen.

**ex:**

class MyThread extends Thread

{

int total=0;

public void run()

{

synchronized(this)

{

System.out.println("Child Thread started calculation");

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

{

total=total+i;

}

System.out.println("Child thread giving notification");

this.notify();

}

}

}

class Test

{

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

{

MyThread t=new MyThread();

t.start();

synchronized(t)

{

System.out.println("Main Thread waiting for updating");

t.wait();

System.out.println("Main -Thread got notification ");

System.out.println(t.total);

}

}

}

DeadLock in java

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

DeadLock will occur in a suitation when one thread is waiting to access

object lock which is acquired by another thread and that thread is waiting

to access object lock which is acquired by first thread.

Here both the threads are waiting release the thread but no body will

release such situation is called DeadLock.

ex:

class Test

{

public static void main(String[] args)

{

final String res1="hi";

final String res2="bye";

Thread t1=new Thread()

{

public void run()

{

synchronized(res1)

{

System.out.println("Thread1: Locking Resource 1");

synchronized(res2)

{

System.out.println("Thread1: Locking Resource2");

}

}

}

};

Thread t2=new Thread()

{

public void run()

{

synchronized(res2)

{

System.out.println("Thread2: Locking Resource 2");

synchronized(res1)

{

System.out.println("Thread1: Locking Resource 1");

}

}

}

};

t1.start();

t2.start();

}

}

Drawbacks of multithreading

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

1)DeadLock

2)Thread Starvation

**Class 50**

Java 8 Features

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

Functional Interface

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

Interface which contains only one abstract method is called functional interface.

It can have any number of default methods and static methods.

It is also known as SAM interface or Single Abstract Method interface.

The main objective of functional interface is to achieve functional programming.

ex:

a = f1()

{

}

f1(f2(){})

{

}

@FunctionalInterface annotation is used to declare functional interface and it is optional.

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1();

}

class B implements A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

}

}

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1();

}

class Test

{

public static void main(String[] args)

{

A a=new A()

{

public void m1()

{

System.out.println("From M1 Method");

}

};

a.m1();

}

}

Lamda Expression

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

Lamda expression introduced in Java 8.

It is used to concise the code.

We can use lamda expression when we have functional interface.

Lamda expression consider as method.

The main objective of lamda expression is to achieve functional programming.

Lamda expression does not allow name, returntype and modifier.

ex:

Java method

-----------

public void m1()

{

System.out.println("M1 Method");

}

Lamda Expression

---------------

()->

{

System.out.println("M1 Method");

};

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1();

}

class Test

{

public static void main(String[] args)

{

A a=()->

{

System.out.println("M1 Method");

};

a.m1();

}

}

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1(int i,int j);

}

class Test

{

public static void main(String[] args)

{

A a=(int i,int j)->

{

System.out.println(10+20);

};

a.m1(10,20);

}

}

ex:

---

@FunctionalInterface

interface A

{

public abstract int m1(int i,int j);

}

class Test

{

public static void main(String[] args)

{

A a=(int i,int j)->

{

return i+j;

};

System.out.println(a.m1(100,200));

}

}

default methods in interface

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

Java provides facility to declare default methods in interface.

If we declare any method in interface and tagged with default keyword is called default method.

It is a non-abstract method.

It can be override.

ex:

---

interface A

{

//abstract method

public abstract void m1();

//default method

default void m2()

{

System.out.println("M2 Method");

}

}

class B implements A

{

public void m1()

{

System.out.println("M1 Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

a.m2();

}

}

ex:

--

interface A

{

//abstract method

public abstract void m1();

//default method

default void m2()

{

System.out.println("M2 Method");

}

}

class B implements A

{

public void m1()

{

System.out.println("M1 Method");

}

public void m2()

{

System.out.println("M2 Method Override");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

a.m2();

}

}

Using default methods of an interface we can achieve multiple inheritance in java.

ex:

---

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

System.out.println("Middle-M1 Method");

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

ex:

--

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

Right.super.m1();

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

ex:

---

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

Left.super.m1();

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

static methods in interface

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

Java provides facility to declare static methods in interface.

If we declare any method in interface and tagged with static keyword is called static method.

It is a non-abstract method.

It can't be override.

ex:

---

interface A

{

static void m1()

{

System.out.println("M1 Method");

}

}

class Test

{

public static void main(String[] args)

{

A.m1();

}

}

Stream API

===========

Stream API introduced in Java 8.

It is used to perform bulk operations on Collections.

If we want to process the objects from Collections we need to use Stream API.

It is present in java.util.stream package.

**Stream Creation**

* **stream()**: Creates a stream from a collection.
* **Arrays.stream():** Creates a stream from an array.
* **Stream.of():** Creates a stream from specified values.(wrapper class /objects)
* **iterate():** Generates an infinite stream by iterating.
* **generate()**: Creates an infinite stream by generating values.

**Intermediate Operations**

* **filter()**: Filters elements based on a predicate.
* **map()**: Transforms each element using a function.
* **flatMap()**: Flattens nested streams into a single stream.
* **distinct()**: Removes duplicate elements from the stream.
* **sorted()**: Sorts the stream elements.
* **limit()**: Limits the stream to a specified number of elements.
* **skip()**: Skips the first specified number of elements.

**Terminal Operations**

* **forEach()**: Performs an action for each element in the stream.
* **collect()**: Accumulates elements into a collection.
* **count()**: Counts the number of elements in the stream.
* **reduce()**: Combines elements into a single value.
* **anyMatch()**: Checks if any elements match a predicate.
* **allMatch()**: Checks if all elements match a predicate.
* **noneMatch()**: Checks if no elements match a predicate.
* **findFirst()**: Retrieves the first element of the stream.
* **findAny()**: Retrieves any element of the stream.

**Collectors**

* **toList()**: Collects elements into a List.
* **toSet()**: Collects elements into a Set.
* **toMap()**: Collects elements into a Map.
* **joining()**: Concatenates elements into a single String.
* **groupingBy()**: Groups elements by a classifier function.
* **partitioningBy()**: Partitions elements into two groups based on a predicate.

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

List<Integer> newList=list.stream().filter(i->i%2==0).collect(Collectors.toList());

System.out.println(newList);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

long count=list.stream().filter(i->i%2!=0).count();

System.out.println(count);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

List<Integer>newList=list.stream().sorted().collect(Collectors.toList());

System.out.println(newList);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

List<Integer> newList=list.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toList());

System.out.println(newList);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

long minimum=list.stream().min((i1,i2)->i1.compareTo(i2)).get();

System.out.println(minimum);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

long maximum=list.stream().max((i1,i2)->i1.compareTo(i2)).get();

System.out.println(maximum);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,7,8,4,1,2,5,5,9,4);

List<Integer>newList=list.stream().distinct().collect(Collectors.toList());

System.out.println(newList);

}

}

forEach() method

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

A forEach() method introduced in Java 8.

It is used to iterate the objects from Collections.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,4,1,2,5,9);

list.forEach(element ->System.out.print(element+" "));

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Map<Integer,String> map=new LinkedHashMap<Integer,String>();

map.put(1,"one");

map.put(2,"two");

map.put(3,"three");

map.forEach((key,value)->System.out.print(key+"="+value+" "));

}

}

Method Reference(::)

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

Method reference introduced in java 8.

Method reference is used to refer method of functional interface.

Method reference is a special type of lambda expression.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,4,1,2,5,9);

list.forEach(System.out::println);

}

}

Interview Question

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

Q) Write a java program to display employee information based on sorting order of employee id?

import java.util.\*;

import java.util.stream.\*;

class Employee

{

private int empId;

private String empName;

private double empSal;

//parameterized constructor

public Employee(int empId,StringempName,doubleempSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

//getter methods

public int getEmpId()

{

return empId;

}

public String getEmpName()

{

return empName;

}

public double getEmpSal()

{

return empSal;

}

}

class Test

{

public static void main(String[] args)

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(104,"Lisa",4000d));

list.add(new Employee(101,"Alan",1000d));

list.add(new Employee(102,"Ben",2000d));

list.add(new Employee(103,"Jessi",3000d));

List<Employee> newList=list.stream().sorted(Comparator.comparingInt(Employee::getEmpId)).collect(Collectors.toList());

newList.forEach(employee ->System.out.println(employee.getEmpId()+" "+employee.getEmpName()+" "+employee.getEmpSal()));

}

}

Q) Write a java program to display employee information based on sorting order of employee name?

import java.util.\*;

import java.util.stream.\*;

class Employee

{

private int empId;

private String empName;

private double empSal;

//parameterized constructor

public Employee(int empId,StringempName,doubleempSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

//getter methods

public int getEmpId()

{

return empId;

}

public String getEmpName()

{

return empName;

}

public double getEmpSal()

{

return empSal;

}

}

class Test

{

public static void main(String[] args)

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(104,"Lisa",4000d));

list.add(new Employee(101,"Alan",1000d));

list.add(new Employee(102,"Ben",2000d));

list.add(new Employee(103,"Jessi",3000d));

List<Employee> newList=list.stream().sorted(Comparator.comparing(Employee::getEmpName)).collect(Collectors.toList());

newList.forEach(employee ->System.out.println(employee.getEmpId()+" "+employee.getEmpName()+" "+employee.getEmpSal()));

}

}

Assignment

==========

Q) Write a java program to display duplicate and unique elements from given array?

input:

1 1 2 3 3 4 5 5 6

output:

duplicate elements : 1 3 5

unique elements : 2 4 6

Java 8 introduced several significant features that enhanced the language and its libraries. Here’s a concise overview of the main methods associated with these features:

### 1. ****Lambda Expressions****

* **Syntax**: (parameters) -> expression or (parameters) -> { statements; }
* **Usage**: Allows you to implement functional interfaces in a clear and concise way.

### 2. ****Functional Interfaces****

* **@FunctionalInterface**: An annotation to indicate an interface with a single abstract method (e.g., Runnable, Callable, Comparator).

### 3. ****Stream API****

* **stream()**: Converts a collection to a stream for processing.
* **filter(Predicate<T> predicate)**: Filters elements based on a condition.
* **map(Function<T, R> mapper)**: Transforms elements using a function.
* **flatMap(Function<T, Stream<R>> mapper)**: Flattens nested streams.
* **distinct()**: Removes duplicates.
* **sorted()**: Sorts elements.
* **limit(long maxSize)**: Limits the number of elements.
* **skip(long n)**: Skips the first n elements.
* **collect(Collector<? super T, A, R> collector)**: Collects elements into a collection.
* **reduce(B identity, BiFunction<B, ? super T, B> accumulator)**: Combines elements into a single result.

### 4. ****Optional Class****

* **Optional<T>**: A container for potentially absent values.
* **of(T value)**: Creates an Optional with a non-null value.
* **empty()**: Creates an empty Optional.
* **ofNullable(T value)**: Creates an Optional that may or may not contain a value.
* **isPresent()**: Checks if a value is present.
* **ifPresent(Consumer<? super T> action)**: Executes an action if a value is present.
* **get()**: Retrieves the value, throws an exception if absent.
* **orElse(T other)**: Returns the value or a default if absent.

### 5. ****Default and Static Methods in Interfaces****

* **default**: Allows methods with a body in interfaces.

java

Copy code

default void defaultMethod() {

// implementation

}

* **static**: Allows static methods in interfaces.

java

Copy code

static void staticMethod() {

// implementation

}

### 6. ****New Date and Time API****

* **LocalDate**: Represents a date without time.
* **LocalTime**: Represents a time without date.
* **LocalDateTime**: Combines date and time.
* **ZonedDateTime**: Represents date and time with time zone.
* **Duration**: Represents a time-based amount of time.
* **Period**: Represents a date-based amount of time.

### 7. ****Method References****

* **ClassName::methodName**: Refers to a method by its name.
  + **Examples**:
    - String::toLowerCase
    - Math::max
    - ArrayList::new

### 8. ****Nashorn JavaScript Engine****

* **ScriptEngineManager**: A new way to execute JavaScript code within Java applications.

### 9. ****CompletableFuture****

* **CompletableFuture<T>**: Represents a future result of an asynchronous computation.
* **supplyAsync(Supplier<U> supplier)**: Asynchronously computes a value.
* **thenApply(Function<? super T,? extends U> fn)**: Transforms the result.
* **thenAccept(Consumer<? super T> action)**: Consumes the result without returning.
* **exceptionally(Function<Throwable, ? extends T> fn)**: Handles exceptions.

### Conclusion

Java 8 introduced functional programming concepts and made the language more expressive and efficient. These features significantly enhance the ability to write cleaner, more readable, and maintainable code.

4o mini

##### You said:

All complete methods in java 8 features

###### ChatGPT said:

ChatGPT

Here’s a detailed overview of key features introduced in Java 8, along with their complete methods, grouped by category.

### 1. ****Lambda Expressions****

Lambda expressions do not have methods per se, but they enable the implementation of functional interfaces.

### 2. ****Functional Interfaces****

Common functional interfaces in java.util.function package:

* **Predicate<T>**: Represents a boolean-valued function.
  + boolean test(T t)
* **Function<T, R>**: Represents a function that accepts one argument and produces a result.
  + R apply(T t)
* **Consumer<T>**: Represents an operation that takes a single input and returns no result.
  + void accept(T t)
* **Supplier<T>**: Represents a supplier of results.
  + T get()
* **UnaryOperator<T>**: A specialization of Function for the case where the input and output types are the same.
  + T apply(T t)
* **BinaryOperator<T>**: A specialization of BiFunction for the case where the input and output types are the same.
  + T apply(T t1, T t2)

### 3. ****Stream API****

Key methods for creating and processing streams:

#### Stream Creation

* **stream()**: Stream<T> stream()
* **Arrays.stream(array)**: Stream<T> stream(T[] array)
* **Stream.of(values)**: static <T> Stream<T> of(T... values)

#### Intermediate Operations

* **filter(Predicate<? super T> predicate)**: Stream<T> filter(Predicate<? super T> predicate)
* **map(Function<? super T, ? extends R> mapper)**: Stream<R> map(Function<? super T, ? extends R> mapper)
* **flatMap(Function<? super T, ? extends Stream<? extends R>> mapper)**: Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>> mapper)
* **distinct()**: Stream<T> distinct()
* **sorted()**: Stream<T> sorted()
* **sorted(Comparator<? super T> comparator)**: Stream<T> sorted(Comparator<? super T> comparator)
* **limit(long maxSize)**: Stream<T> limit(long maxSize)
* **skip(long n)**: Stream<T> skip(long n)

#### Terminal Operations

* **forEach(Consumer<? super T> action)**: void forEach(Consumer<? super T> action)
* **collect(Collector<? super T, A, R> collector)**: R collect(Collector<? super T, A, R> collector)
* **count()**: long count()
* **reduce(T identity, BinaryOperator<T> accumulator)**: T reduce(T identity, BinaryOperator<T> accumulator)
* **reduce(BinaryOperator<T> accumulator)**: Optional<T> reduce(BinaryOperator<T> accumulator)
* **anyMatch(Predicate<? super T> predicate)**: boolean anyMatch(Predicate<? super T> predicate)
* **allMatch(Predicate<? super T> predicate)**: boolean allMatch(Predicate<? super T> predicate)
* **noneMatch(Predicate<? super T> predicate)**: boolean noneMatch(Predicate<? super T> predicate)
* **findFirst()**: Optional<T> findFirst()
* **findAny()**: Optional<T> findAny()

### 4. ****Optional Class****

Methods in the Optional<T> class:

* **Optional<T> of(T value)**: Creates an Optional with a non-null value.
* **Optional<T> empty()**: Creates an empty Optional.
* **Optional<T> ofNullable(T value)**: Creates an Optional that may or may not contain a value.
* **boolean isPresent()**: Checks if a value is present.
* **void ifPresent(Consumer<? super T> action)**: Executes an action if a value is present.
* **T get()**: Retrieves the value, or throws an exception if absent.
* **T orElse(T other)**: Returns the value or a default if absent.
* **T orElseGet(Supplier<? extends T> other)**: Returns the value or computes a default if absent.
* **Optional<T> or(Supplier<? extends Optional<? extends T>> supplier)**: Returns the value or computes a new Optional if absent.
* **boolean equals(Object obj)**: Checks if the Optional is equal to another object.
* **int hashCode()**: Returns the hash code of the Optional.
* **String toString()**: Returns a string representation of the Optional.

### 5. ****Default and Static Methods in Interfaces****

You can add default and static methods directly to interfaces:

* **default void defaultMethod()**: Implementation within an interface.
* **static void staticMethod()**: Static method that can be called without an instance.

### 6. ****New Date and Time API****

Classes and methods in the java.time package:

* **LocalDate**: Represents a date without time.
  + static LocalDate now()
  + static LocalDate of(int year, int month, int dayOfMonth)
  + LocalDate plusDays(long daysToAdd)
  + LocalDate minusDays(long daysToSubtract)
* **LocalTime**: Represents a time without a date.
  + static LocalTime now()
  + static LocalTime of(int hour, int minute)
  + LocalTime plusHours(long hoursToAdd)
  + LocalTime minusMinutes(long minutesToSubtract)
* **LocalDateTime**: Combines date and time.
  + static LocalDateTime now()
  + static LocalDateTime of(LocalDate date, LocalTime time)
  + LocalDateTime plusDays(long daysToAdd)
  + LocalDateTime minusMinutes(long minutesToSubtract)
* **ZonedDateTime**: Represents date and time with time zone.
  + static ZonedDateTime now()
  + static ZonedDateTime of(LocalDateTime dateTime, ZoneId zone)
* **Duration**: Represents a time-based amount of time.
  + static Duration ofSeconds(long seconds)
  + static Duration between(Temporal startInclusive, Temporal endExclusive)
* **Period**: Represents a date-based amount of time.
  + static Period ofYears(int years)
  + static Period between(LocalDate startDateInclusive, LocalDate endDateExclusive)

### 7. ****Method References****

Method references do not have explicit methods but are used as shorthand for lambda expressions. Common types include:

* **Static method reference**: ClassName::staticMethodName
* **Instance method reference of a particular object**: instance::instanceMethodName
* **Instance method reference of an arbitrary object of a particular type**: ClassName::instanceMethodName
* **Constructor reference**: ClassName::new

### 8. ****Nashorn JavaScript Engine****

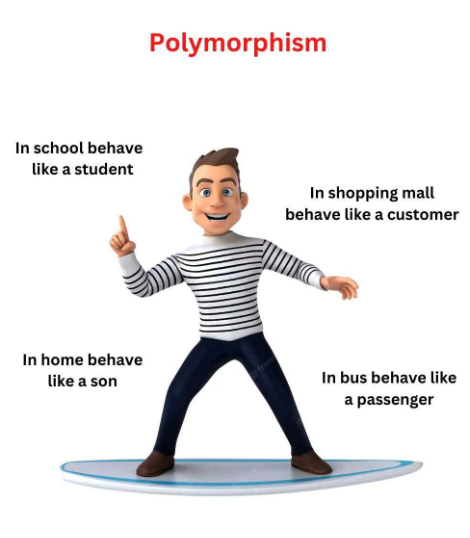
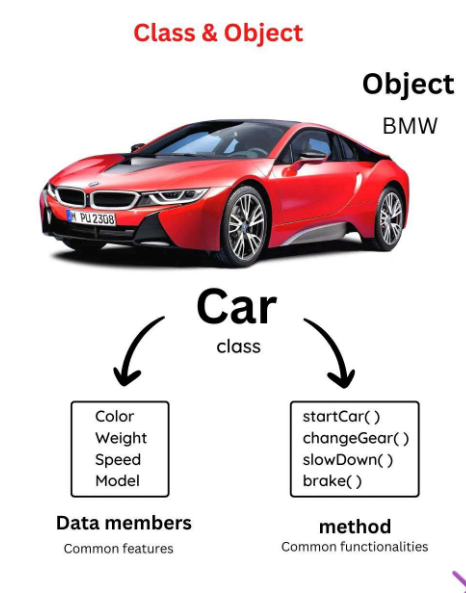
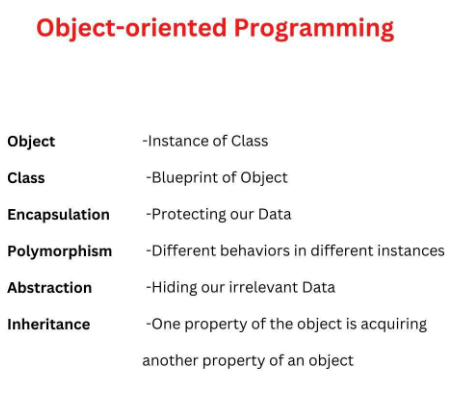
Key classes and methods for scripting:

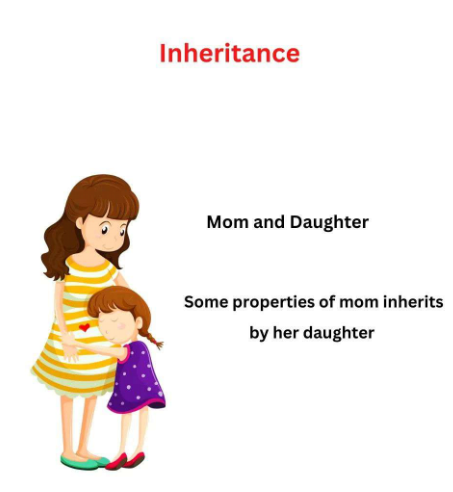
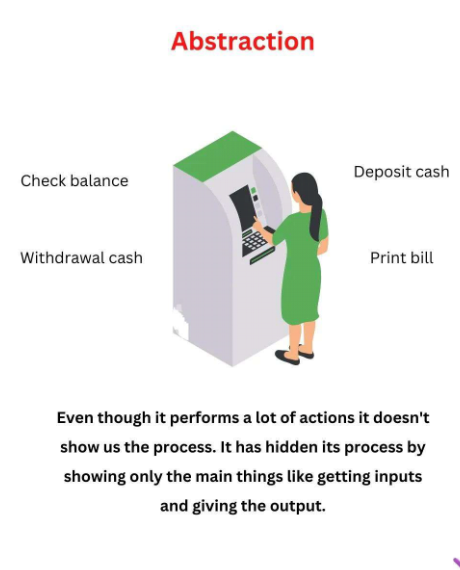
* **ScriptEngineManager**: Manages script engines.
  + ScriptEngine getEngineByName(String shortName)
* **ScriptEngine**: Represents a scripting engine.
  + Object eval(String script)
  + void put(String name, Object value)

### 9. ****CompletableFuture****

Key methods in the CompletableFuture<T> class:

* **static CompletableFuture<T> supplyAsync(Supplier<T> supplier)**: Starts a new asynchronous computation.
* **CompletableFuture<T> thenApply(Function<? super T,? extends U> fn)**: Transforms the result.
* **CompletableFuture<Void> thenAccept(Consumer<? super T> action)**: Consumes the result without returning.
* **CompletableFuture<T> exceptionally(Function<Throwable, ? extends T> fn)**: Handles exceptions.
* **CompletableFuture<Void> allOf(CompletableFuture<?>... futures)**: Returns a CompletableFuture that completes when all the given futures complete.
* **CompletableFuture<T> anyOf(CompletableFuture<?>... futures)**: Returns a CompletableFuture that completes when any of the given futures complete.





Class 31:-

OOPS

=====

OOPS stands for Object Oriented Programming System/Structure.

OOPS allows us to deals with real world entities using programming language.

We have following important features present in oops.

ex:

class

object

Abstraction

Encapsulation

Inheritance

Polymorphism

class

==========

A class is a blue print of an object.

A class is a logical entity.

A class is a collection of objects.

We can declare a class as follow.

syntax:

optional

|

modifier class class\_name <extends> parent\_classname

<implements> interface\_name

{

-

- //set of objects

-

}

A class will accept following modifiers.

ex:

default

public

abstract

final

Q) What is difference between default class and public class?

default class public class

--------------- -------------

To declare default class we should not To declare public class we should use

use any modifier. public modifier.

ex: ex:

class A public class A

{ {

} }

If we declare any class as default then we If we declare any class as public then we can

can access that class within the package. access that within the package and outside the

package.

Q) What is difference between abstract and final class?

abstract class final class

------------------ ---------

To declare abstract class we should use To declare final class we should use

abstract modifier. final modifier.

ex: ex:

abstract class A final class A

{ {

} }

Object creation is not possible. Object creation is possible.

Child creation is possible. Child creation is not possible.

Class 32:-

========

object

It is a outcome of a blue print.

It is a instance of a class.

Here instance means allocating memory for our data members.

It is a physical entity.

It is a collection of properties and behaviours.

ex:

Dog

|

|-----------------------------------------------|

Properties Behaviours

> Name > Barking

> Color > Eating

> Age > Sleeping

> Weight and etc.

> Height

and etc.

Memory space will be created when we create an object.

We can declare object as follow.

ex:

operator

|

Test t = new Test();

| | |

classname reference\_var constructor\_name

It is possible to create more then one object in a single class.

ex:

----

class Test

{

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test();

Test t3=new Test();

System.out.println(t1.hashCode());

System.out.println(t2.hashCode());

System.out.println(t3.hashCode());

System.out.println(t1);//Test@Hexadecimal

System.out.println(t2.toString());

System.out.println(t3.toString());

}

}

Q) What is hashcode in java?

Every object JVM will create a unique identification number i.e hash code.

In order to read the hash code of an object we need to use hashCode() method.

A hashCode() method present in Object class.

Diagram: class32.1

Q) What is toString()?

Whenever we are trying to display object reference directly or indirectly toString() method

will be executed.

A toString() method is also present in Object class.

Q) What is the difference between class and object?

class object

---------- -------

To declare a class we will use class keyword. To declare object we will use new keyword.

It is a collection of objects. It is a collection of properties and behaviours.

It is a blueprint of an object. It is a outcome of a blue print.

It is a logical entity. It is a physical entity.

It can't manipulate. It can manipulate.

It does not allocate the memory. It allocates the memory.

It declares once. It declares many times.

Data Hiding

===========

Data hiding is a process of hiding object data from the outsiders.

Using private modifier we can implements data hiding concept.

The main objective of data hiding is to provide security.

ex:

---

class Account

{

private double balance=5000d;

}

class Student

{

public static void main(String[] args)

{

Account account=new Account();

System.out.println(account.balance);

}

}

Abstraction

===========

Hiding internal implementation and highlighting the set of services is called abstraction.

Using abstract classes and interfaces we can implements abstraction.

The best example of abstraction is GUI ATM machine because bank people will hide internal implementation and highlights the set of services like banking, withdrawl, mini statement and etc.

The main advantages of abstraction are.

1) It gives security because it will hide internal implementation.

2) Enhancement becomes more easy because without effecting enduser they can perform any changes

in our internal system.

3) It provides flexibility to the end user to use the system.

4) It improves maintainability of an application.

Class 33:-

Encapsulation

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

The process of encapsulating or grouping variables and it's associate methods in a single entity is called encapsulation.

Diagram: class33.1

A class is said to be encapsulated class if it supports data hiding and abstraction.

Abstraction is used to hide the data and encapsulation is used to protect the data.

In encapsulation , for every variable we need to declare setter and getter methods.

Diagram: class33.2

The main advantages of encapsulation are.

1) It gives security.

2) Enhancements becomes more easy.

3) It provides flexibility to the end user to use the system.

4) It improves maintainability of an application.

The main disadvantage of encapsulation is , it will increase the length of our code and slow down the execution process.

ex:

----

class Student

{

//current class variables

private int studId;

private String studName;

private double studFee;

//setter methods

public void setStudId(int studId)

{

this.studId=studId;

}

public void setStudName(String studName)

{

this.studName=studName;

}

public void setStudFee(double studFee)

{

this.studFee=studFee;

}

//getter methods

public int getStudId()

{

return studId;

}

public String getStudName()

{

return studName;

}

public double getStudFee()

{

return studFee;

}

}

class Test

{

public static void main(String[] args)

{

Student s=new Student();

s.setStudId(101);

s.setStudName("Alan");

s.setStudFee(1000d);

System.out.println("Student Id :"+s.getStudId());

System.out.println("Student Name :"+s.getStudName());

System.out.println("Student Fee :"+s.getStudFee());

}

}

Q) What is tightly encapsulated class?

A class is said to be tightly encapsulated class if and only if all variables of that class declared as private. Here we don't need to check these variables having setter and getter method or not.

ex:

class A

{

private int i;

private int j;

}

What is the difference between Abstraction and Encapsulation?

Abstraction Encapsulation

-------------- -----------

Hiding internal implementation and The process of encapsulating or grouping

highlighting the set of services is called variables and its associate methods in a

abstraction. single entity is called encapsulation.

It is used to hide the data. It is used to protect the data.

Using abstract classes and interfaces we Using access modifiers we can implements

can implements abstraction. encapsulation.

It is a process of gaining the information. It is a process of containing the information.

It solves an issue at design level. It solves an issue at implementation level.

Q)What is the difference between POJO class and Java Bean class?

POJO Java Bean

-------- -----------

It can’t be serialized. It can be serialized.

Fields can have any visibility. Fields can have only private visibility.

There may or may not have 0-arg constructor. It must have 0-argument constructor.

It does not extends any other class. It can extends.

It does not implement any other interface. It can implements.

It does not use any outside annotation. It uses outside annotation.

Is-A relationship

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

Is-A relationship is also known as inheritance.

Using extends keyword we can implements Is-A Relationship.

The main objective of Is-A relationship is to provide reusability.

ex:

---

class Parent

{

public void methodOne()

{

System.out.println("Method One");

}

}

class Child extends Parent

{

public void methodTwo()

{

System.out.println("Method Two");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.methodOne();

Child c=new Child();

c.methodOne();

c.methodTwo();

Parent p1=new Child();

p1.methodOne();

//Child c1=new Parent();

}

}

Conclusion

----------

Whatever parent having properties by default it comes to child. But whatever child having properties it never goes back to parent.

A parent reference can hold child object but child reference can't hold parent object.

Inheritance

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

Inheritance is a mechanism where we will derived a class in the presence of existing class.

Inheritance is a mechanism where one class will inherit the properties of another class.

The main objective of inheritance is to provide reusability.

Diagram: class33.3

We have following list of inheritance in java.

1) Single Level inheritance

2) Multi Level inheritance

3) Multiple inheritance

4) Hirarchical inheritance

5) Hybrid inheritance

1) Single Level inheritance

-----------------------------

If we derived a class in the presence of one base class is called single level inheritance.

Diagram:

A (parent class/ super class/ base class)

|

|

|

B (child class / sub class / derived class)

ex:

----

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

}

}

ex:

----

class A

{

int i=10;

}

class B extends A

{

int j=20;

}

class Test

{

public static void main(String[] args)

{

A a=new A();

System.out.println(a.i);//10

B b =new B();

System.out.println(b.i+" "+b.j);//10 20

}

}

2) Multi Level inheritance

------------------------

If a class is derived by using one base class and that class is derived from another base class is called multi level inheritance.

ex:

---

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class C extends B

{

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

C c=new C();

c.m1();

c.m2();

c.m3();

}

}

3) Multiple inheritance

------------------------

In java, a class can't extends more then one class simultenously because java does not support multiple inheritance.

ex:

class A

{

}

class B

{

}

class C extends A,B --> invalid

{

}

Interface can extends more then one interface so we can achieve multiple inheritance concept through interfaces.

ex:

interface A

{

}

interface B

{

}

interface extends A,B --> valid

{

}

If our class does not extends any other class then it is a direct child class of Object class.

ex: diag:

class A Object

{ |

|

} A

If our class extends some other class then it is a indirect child class of Object class.

ex: diag:

class A Object

{ |

} |

class B extends A A

{ |

} |

B

Java does not support cyclic inheritance.

ex:

class A extends B

{

}

class B extends A

{

}

Q) Why java does not support multiple inheritance?

There may a chance of raising ambiguity problem that's why java does not support multiple inheritance.

ex:

p1.m1() p2.m1()

|-------------------------------------|

|

c.m1()

Interview Question

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

Q) What is jagged array?

Jagged array is also known as array of arrays.

It is a multi dimensional array where each row having different size.

Diagram: class33.4

ex:

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int[][] arr=new int[3][];

arr[0]=new int[3];

arr[1]=new int[2];

arr[2]=new int[4];

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

{

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

{

System.out.println("Enter the element of arr["+i+"]["+j+"] :");

arr[i][j]=sc.nextInt();

}

}

//reading

for(int[] a:arr)

{

for(int i:a)

{

System.out.print(i+" ");

}

//new

System.out.println();

}

}

}

Class 34:-

4) Hierarchical inheritance

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

If we derived multiple class in the presence of one base class is called hierarchical inheritance.

Diagram:

A

|

|-------------------------------|

B C

ex:

---

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class C extends A

{

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

C c=new C();

c.m1();

c.m3();

}

}

5) Hybrid inheritance

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

It is a combination of more then one inheritance.

Java does not support hybrid inheritance.

Diagram:

A

|

|-------------------------------|

B C

|-------------------------------|

|

D

Has-A relationship

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

Has-A relationship is also known as composition and aggregation.

There is no specific keyword to implements Has-A relationship but mostly we will use new operator.

The main objective of Has-A relationship is to provide reusability.

Has-A relationship will increase dependency between two components.

ex:

class Trainer

{

}

class Course

{

Trainer t=new Trainer();

-

-

}

ex:

---

class Ihub

{

public String courseName()

{

return "Full Stack Java + AWS";

}

public double courseFee()

{

return 30000d;

}

public String trainerName()

{

return "Niyaz Sir";

}

}

class Usha

{

public void getCourseDetails()

{

Ihub i=new Ihub();

System.out.println("Course Name :"+i.courseName());

System.out.println("Course Fee :"+i.courseFee());

System.out.println("Trainer Name :"+i.trainerName());

}

}

class Student

{

public static void main(String[] args)

{

Usha u=new Usha();

u.getCourseDetails();

}

}

composition

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

Without existing container object there is no chance of having contained object then the relationship between container and contained object is called composition which is strongly association.

Diagram: class34.1

aggregation

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

Without existing container object there is a chance of having contained object then the relationship between container and contained object is called aggregation which is loosely association.

Diagram: class34.2

Method overloading

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

Having same method name with different parameters/signatures in a single class is called method overloading.

All the methods present in a class are called overloaded methods.

Method overloading will reduce complexity of the programming.

Method resolution will taken care by a compiler based on reference type.

ex:

---

class MeeSeva

{

//overloaded methods

public void search(int voterId)

{

System.out.println("Details Found via voterId");

}

public void search(String houseNo)

{

System.out.println("Details Found via houseNo");

}

public void search(long aadharNo)

{

System.out.println("Details Found via aadharNo");

}

}

class Test

{

public static void main(String[] args)

{

MeeSeva ms=new MeeSeva();

ms.search(101);

ms.search("1-4-6/3");

ms.search(1234L);

}

}

Method overriding

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

Having same method name with same parameters into two different classes is called method overriding.

Methods which are present in parent class are called overridden methods.

Methods which are present in child class are called overridding methods.

Method resolution will taken care by JVM based on runtime object.

ex:

---

class Parent

{

public void property()

{

System.out.println("Cash+Gold+Land+House");

}

public void marry()

{

System.out.println("Trisha");

}

}

class Child extends Parent

{

public void marry()

{

System.out.println("Rashmika");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

p.marry(); // Trisha

Child c=new Child();

c.property(); // Cash+Gold+Land+House

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // Cash+Gold+Land+House

p1.marry(); // Rashmika

}

}

If we declare any method as final then overridding of that method is not possible.

ex:

---

class Parent

{

//overridden methods

public void property()

{

System.out.println("Cash+Gold+Land+House");

}

public final void marry()

{

System.out.println("Trisha");

}

}

class Child extends Parent

{

//overridding methods

public void marry()

{

System.out.println("Rashmika");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

p.marry(); // Trisha

Child c=new Child();

c.property(); // Cash+Gold+Land+House

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // Cash+Gold+Land+House

p1.marry(); // Rashmika

}

}

If parent don't want to share any properties to child class then we need to declare variables and methods as private.

Private methods can't be override.

ex:

class Parent

{

//overridden methods

private void property()

{

System.out.println("father property");

}

}

class Child extends Parent

{

//overridding methods

public void property()

{

System.out.println("child property");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

Child c=new Child();

c.property(); // Cash+Gold+Land+House

}

}

Q) Can we overload main method in java?

Yes, we can overload main method in java but JVM always execute main method with String[] parameter only.

ex:

class Test

{

public static void main(String[] args)

{

System.out.println("String arg method");

}

public static void main(int[] iargs)

{

System.out.println("int arg method");

}

}

Method Hiding

===========

Method hiding is exactly same as method overriding with following differences.

Method Overriding Method Hiding

------------------- ---------------

All methods present in method overriding All methods present in method hiding must be

must be non-static. static.

Method resolution will taken care by Method resolution will taken care by compiler based

JVM based on runtime object. on reference type.

It is also known as dynamic polymorphism, It is also known as static polymorphism, compile time

runtime polymorphism or late binding. polymorphism or late binding.

class Parent

{

public static void property(){

System.out.println("Cash+Gold+Land+House");

}

public static void marry(){

System.out.println("Trisha");

}

}

class Child extends Parent{

public static void marry(){

System.out.println("Rashmika");

}

}

class Test {

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

p.marry(); // Trisha

Child c=new Child();

c.property(); // Cash+Gold+Land+House

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // Cash+Gold+Land+House

p1.marry(); // Trisha

}

}

Q) Can we override main method in java?

No, we can't override main method in java because it is static.

Class 35:-

Polymorphism

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

polymorphism has taken from Greek word.

Here poly means many and morphism means forms.

The ability to represent in different forms is called polymorphism.

The main objective of polymorphism is to provide flexibility.

Diagram: class35.1

In java we have two types of polymorphism.

1) Compile time polymorphism / Static polymorphism / Early Binding

2) Runtime polymorphism / Dynamic polymorphism / Late Binding

1) Compile time polymorphism

---------------------------

A polymorphism which exhibits(display) at compile time is called compile time polymorphism.

ex:

Method overloading

Method Hiding

2) Runtime polymorphism

-----------------------

A polymorphism which exhibits at runtime is called runtime polymorphism.

ex:

Method Overriding

Diagram: class35.2

Constructor

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

A constructor is a special method which is used to initialized an object.

Having same name as class name is called constructor.

A constructor will execute when we create an object.

A constructor does not allow any returntype.

A constructor will accept following modifiers.

ex:

default

public

private

protected

In java, constructors are divided into two types.

1) Userdefined constructor

2) Default constructor

1) Userdefined constructor

--------------------------

Constructor which is created by the programmer or user based on the application requirement is called userdefined constructor.

It is classified into two types.

i) Zero Argument constructor

ii) Parameterized constructor

i) Zero Argument constructor

-----------------------------

Suppose if we are not passing any argument to userdefined constructor then that constructor is called zero argument constructor.

ex:

---

class Test

{

Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

System.out.println("main-method");

}

}

o/p:

main-method

ex:

---

class Test

{

public Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

System.out.println("main-method");

Test t=new Test();

Test t=new Test();

}

}

ex:

---

class Test

{

private Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

Test t1=new Test();

System.out.println("main-method");

Test t2=new Test();

}

}

o/p:

0-arg const

main-method

0-arg const

ex:

--

class Test

{

protected Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

Test t1=new Test();

System.out.println("main-method");

Test t2=new Test();

}

}

o/p:

0-arg const

main-method

0-arg const

ii) Parameterized constructor

-----------------------------

Suppose if we are passing atleast one argument to userdefined constructor then that constructor is called parameterized constructor.

ex

---

class Employee

{

//current class variables

private int empId;

private String empName;

private double empSal;

public Employee(int empId,String empName,double empSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

public void getEmployeeDetails()

{

System.out.println("Employee Id :"+empId);

System.out.println("Employee Name :"+empName);

System.out.println("Employee Salary :"+empSal);

}

}

class Test

{

public static void main(String[] args)

{

Employee e=new Employee(101,"Alan Morries",1000d);

e.getEmployeeDetails();

}

}

2) Default constructor

------------------------

It is a compiler generated constructor for every java program where we are not defining atleast

zero argument constructor.

To see the default constructor we need use below command.

ex:

javap -c Test

Diagram: class35.3

Constructor Overloading

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

Having same constructor name with different parameters or signatures in a single class is called constructor overloading.

eX:

---

class Test

{

Test()

{

System.out.println("0-arg const");

}

Test(int i)

{

System.out.println("int-arg const");

}

Test(double d)

{

System.out.println("double-arg const");

}

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test(10);

Test t3=new Test(10.5d);

}

}

this keyword

===========

A this keyword is a java keyword which is used to refer current class object reference.

We can utility this keyword in following ways.

1) To refer current class variables

2) To refer current class methods

3) To refer current class constructors

1) To refer current class variables

-----------------------------------

class A

{

int i=10;

int j=20;

A(int i,int j)

{

System.out.println(this.i+" "+this.j); // 10 20

System.out.println(i+" "+j); // 100 200

}

}

class Test

{

public static void main(String[] args)

{

A a=new A(100,200);

}

}

2) To refer current class methods

------------------------------

class A

{

public void m1()

{

System.out.println("M1-Method");

this.m2();

}

public void m2()

{

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

}

}

3) To refer current class constructor

-------------------------------------

class A

{

A()

{

System.out.println("0-arg const");

}

A(int i)

{

this();

System.out.println("int-arg const");

}

A(double d)

{

this(10);

System.out.println("double-arg const");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A(10.5d);

}

}

super keyword

===========

A super keyword is a java keyword which is used to refer super class object reference.

We can utility super keyword in following ways.

1) To refer super class variables

2) To refer super class methods

3) To refer super class constructors

1) To refer super class variables

----------------------------------

class A

{

int i=10;

int j=20;

}

class B extends A

{

int i=100;

int j=200;

B(int i,int j)

{

System.out.println(i+" "+j); // 1000 2000

System.out.println(this.i+" "+this.j); // 100 200

System.out.println(super.i+" "+super.j); //10 20

}

}

class Test

{

public static void main(String[] args)

{

B b=new B(1000,2000);

}

}

2) To refer super class methods

--------------------------------

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

super.m1();

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

B b=new B();

b.m2();

}

}

API

====

API stands for Application Programming interface.

It is a base for the programmer to develop software applications.

API is a collection of packages.

In java, API is divided into three types.

1) Predefined API

-------------

Built-in API is called predefined API.

ex:

https://docs.oracle.com/javase/8/docs/api/

2) Userdefined API

----------------

API which is created by the user based on the application requirements is called

userdefined API.

3) Third party API

-------------

API which is given by third party vendor is called third party API.

ex:

JAVAZOOM API

iText API

and etc.

Interview Question

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

Q) write a java program to display sub array equals to given sum?

input:

arr = 1 2 3 7 6

sum = 12

output:

2 3 7

approach1

---------

class Test

{

public static void main(String[] args)

{

int[] arr ={1,2,3,7,6};

int sum = 12;

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

{

int currentSum=0;

for(int j=i;j<arr.length;j++)

{

currentSum+=arr[j];

if(currentSum==sum)

{

for(int k=i;k<=j;k++)

{

System.out.print(arr[k]+" ");

}

}

}

}

}

}

approach2

----------

class Test

{

public static void main(String[] args)

{

int[] arr ={1,2,3,7,5};

int sum = 12;

int cnt=0;

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

{

int currentSum=0;

for(int j=i;j<arr.length;j++)

{

currentSum+=arr[j];

if(currentSum==sum)

{

cnt=1;

for(int k=i;k<=j;k++)

{

System.out.print(arr[k]+" ");

}

}

}

if(cnt==1)

{

break;

}

}

}

}

Class36:-

Interface

=========

Interface is a collection of zero or more abstraction methods.

Abstract methods are incomplete methods because they ends with semicolon and does not have any body.

ex

void m1();

It is not possible to create object for interfaces.

To write the implementation of abstract methods we will use implementation class.

It is possible to create object for implementation class because it contains method with body.

By default every abstract method is a public and abstract.

ex:

public abstract void m1();

Interface contains only constants i.e public static final.

We can declare interface as follow.

syntax:

-----

interface <interface\_name>

{

-

- //abstract methods

- //constants

-

}

If we know service requirement specification then we need to use interface.

Diagram: class36.1

ex:

----

interface A

{

//abstract method

public abstract void m1();

}

class B implements A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

}

}

ex:

---

interface A

{

//abstract method

public abstract void m1();

}

class Test

{

public static void main(String[] args)

{

A a=new A()

{

public void m1()

{

System.out.println("From M1 Method");

}

};

a.m1();

}

}

If interface contains four methods then we need to override all methods otherwise we will get compile time error.

ex:

---

interface A

{

public abstract void show();

public void display();

abstract void see();

void view();

}

class B implements A

{

public void show()

{

System.out.println("Show Method");

}

public void display()

{

System.out.println("Display Method");

}

public void see()

{

System.out.println("See Method");

}

public void view()

{

System.out.println("View Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.show();

a.display();

a.see();

a.view();

}

}

A class can't extends more then one class simultenously.

But interface can extends more then one interface.

ex:

---

interface A

{

void m1();

}

interface B

{

void m2();

}

interface C extends A,B

{

void m3();

}

class D implements C

{

public void m1()

{

System.out.println("M1-Method");

}

public void m2()

{

System.out.println("M2-Method");

}

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

C c=new D();

c.m1();

c.m2();

c.m3();

}

}

A class can implements more then one interface.

ex:

---

interface Father

{

float HT=6.2f;

void height();

}

interface Mother

{

float HT=5.8f;

void height();

}

class Child implements Father,Mother

{

public void height()

{

float height=(Father.HT+Mother.HT)/2;

System.out.println("Child Height is ="+height);

}

}

class Test

{

public static void main(String[] args)

{

Child c=new Child();

c.height();

}

}

Note:

------

Interface is a collection of abstract methods, default methods and static methods.

Marker interface

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

Interface which does not have methods and constants is called marker interface.

In general, empty interface is called marker interface.

By using marker interface we will get some ability to do.

We have following list of marker interfaces.

ex:

Serializable

Cloneable

Remote

and etc.

ex:

class Student implements java.io.Serializable

{

private int studId;

public void setStudId(int studId)

{

this.studId=studId;

}

public int getStudId()

{

return studId;

}

}

Abstract class

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

Abstract class is a collection of zero or more abstract methods and concrete methods.

A abstract keyword is applicable for classes and methods but not for variables.

It is not possible to create object for abstract class.

To write the implementation of abstract methods we will use sub classes.

By default every abstract method is a public and abstract.

Abstract class contains only instance variables.

syntax:

------

abstract class <class\_name>

{

-

- //abstract method

- //concrete method

- //instance variable

-

}

If we know partial implementation then we need to use abstract class.

ex:

---

abstract class Plan

{

//instance variable

protected double rate;

//abstract method

public abstract void getRate();

//concrete method

public void calculateBill(int units)

{

System.out.println("Total Units :"+units);

System.out.println("Total Bill :"+ units\*rate);

}

}

class DomesticPlan extends Plan

{

public void getRate()

{

rate=2.5d;

}

}

class CommercialPlan extends Plan

{

public void getRate()

{

rate=5.0d;

}

}

class Test

{

public static void main(String[] args)

{

DomesticPlan dp=new DomesticPlan();

dp.getRate();

dp.calculateBill(250);

CommercialPlan cp=new CommercialPlan();

cp.getRate();

cp.calculateBill(250);

}

}

Q) What is the difference between interface and abstract class?

Interface Abstract class

------------------ ---------------

To declare interface we will use interface To declare abstract class we will use abstract

keyword. keyword.

It is a collection of abstract methods, default It is a collection of abstract methods and concrete

methods and static methods. methods.

It contains constants. It contains instance variables.

We can achieve multiple inheritance. We can't achieve multiple inheritance.

We will use implementation class to write the We will use sub class to write the logic for

logic for abstract methods. abstract methods.

If we know only specification then we need If we know partial implementation then we need to use

to use interface. abstract class.

Abstraction

===========

Hiding internal implementation and highlighting the set of services is called abstraction.

Using abstract class and interfaces we can implements abstraction.

ex:

abstract class Animal

{

//abstract method

public abstract void makeSound();

}

class Cat extends Animal

{

public void makeSound()

{

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

}

}

class Test

{

public static void main(String[] args)

{

Cat c=new Cat();

c.makeSound();

}

}

**core Java**

*1. What is Java?*

*2. Features of Java?*

*3. JDK vs JRE vs JVM?*

*4. Types of Memories in java?*

*5. What is JIT compiler?*

*6. What is Garbage Collector and how many ways we can call garbage collector?*

*7. Types of class loaders in java?*

*8. Explain main method in java?*

*9. Is java purely object oriented or not?*

*10. What is Object class?*

*11. What is hash code in java?*

*12. What is class and give me one real time example?*

*13. What is object and give me one real time example?*

*14. What is Abstraction and give me one real time example?*

*15. What is Encapsulation with example?*

*16. Method Overloading vs Method Overriding?*

*17. What is polymorphism and types of polymorphism?*

*18. What is API?*

*19. What is package?*

*20. What is singleton class?*

*21. What is constructor?*

*22. What is default constructor in java and how to see the default constructor in java?*

*23. Interface vs Abstract class?*

*24. this keyword vs super keyword?*

*25. What is native method in java?*

*26. What is String?*

*27. length vs length() method?*

*28. StringBuffer vs StringBuilder?*

*29. Difference between == and .equals() method?*

*30. Exception vs Error?*

*31. What is Exception and types of exceptions?*

*32. Difference between final, finally and finalize in java?*

*33. Arrays vs Collection?*

*34. What is Anonymous array?*

*35. What is Collection interface?*

*36. What is inheritance and types?*

*37. Why java does not support multiple inheritance?*

*38. List vs Set interface?*

*39. ArrayList vs Vector?*

*40. HashSet vs LinkedHasSet?*

*41. HashSet vs TreeSet?*

*42. What is Map interface?*

*43. HashMap vs LinkedHashMap?*

*44. HashMap vs TreeMap?*

*45. HashMap vs Hashtable?*

*46. Types of cursors in java?*

*47.Comparable vs Comparator interface?*

*48. What is multithreading?*

*49. Thread vs Process?*

*50. In how many ways we can create a thread in java?*

*51. What is thread scheduler?*

*52. Life cycle of a thread?*

*53. What is Daemon thread in java?*

*54. What is Deadlock in java?*

*55. What is inter-thread communication?*

*56. What is synchronization?*

*57. What is Functional interface?*

*58. What is marker interface?*

*59. What is lamda expression?*

*60. What is Stream API?*

*61. How to achieve multiple inheritances in java?*