Arrays

Int a[] = {1,2,3,4}; correct

Int a[4]= { 1,2,3,4}; wrong ( use new operator)

Int a [];

a[0]=1, a[1]=2, a[2]=3; correct

int a[] = new int[4];

int a[] = new int[4];

int a[] = new int[4];

a[0]=1, a[1]=2, a[2]=3; correct

int a[] = new int[] {1,2,3,4};correct (Anonymous Array)

int a[] = new int[4] {1,2,3,4}; wrong

int a[] = new int[-4]; (NegativeArraySIze Exception)

for each statement

for(int a1 : a)

{

SOP(a) ; //only for printing purpose

}

Condition for for each

Variable should be declared inside the for each of same data type

Single dimension should be stored in a variable

Two dimension stored in one dimension

String

toString --🡪 converts object to String

char charAt(int loc) 🡪 returns Single character

void getChars(int start, int end. Char[] buf, int targetstart)

char[] toCharArray -🡪 convert string to charcter array

byte[] getBytes()

Boolean startsWith(String)

Boolean endsWith(String)

Int length()

Boolean equals(String s)

Checks the equality of content by considering the case

Boolean equalsIgnorecase(String s)

==(equal versus)

Checks the equality of object reference

String substring(int start)

String substring(int start, int end) start to end-1

Int compareTo (Sting s)

Compare two string and sort

Equal – 0, greater than – 1 , less than - -1

Int compareToIgnoreCase (String s)

Int indexOf(char c)

First occurrence of character in given string

Int lastIndexOf( Char c)

Last occurrence of character in given string

S1.indexof(“the”)

S1.indexOf(‘t’, 10); //after 10 first ‘t’

S1.indexOf(“the”,60); //before 60 first t

If no character return -1

String toUppercase()

String toLowercase()

String trim() trim leading and ending spaces

String concat(String s)

String replace(char ori, char replace)

Trim and concat doesn’t affect the original string…bcoz String is immutable

String[] split(String delimiter)

String s18=”one-two-three”;

String s19[] = s18.split(“-“);

for(String s20:s19)

{

System.out.println(s20);

}

For .(dot) delimitor

\\.

For \* delimitor

\\\*

Splitting based on words

String s1= “concession | concillation | (give and take)”;

| --- Single word

() ------ group of words

\\ ----- for . , \*

Static String format (String format, String value)

Format the string and print using System.out.println()

System.out.println(String.format(“|%5d|”,4));

% + - 0 w c b

+ ----- pad before , - -----------🡪 pad after , 0 ---- pad with 0

Boolean regionMatches(Boolean ignorecase , int start. String value, int targetstart, int howmany);

Match a part of string

**Wrapper class**

Class that supports primitive data type to perform operation on that data type

All wrapper classes are also immutable

int ---- Integer

byte ---- Byte

char --- Char

boolean – Boolean

**Integer**

To perform operation on int datatype

Constructors

1. Integer(int)

Integer I =new Integer(42);

SOP(i);

// output 42

Predefined class already overrriden toString method

Integer I =new Integer(“42”); // This is correct

Integer i3 = new Integer(“abc”);

// Number format exception

Methods

Public static int parseInt(String s)

//Convert string to datatype

Public static int parseInt(String s, int radix)

// radix convert the value to base 5

String s = “42”;

Integer I = Integer.parseInt(s,5);

Convert 42 to base 5 --🡪 o/p :22

Int intValue()

//Convert wrapper class to datatype

Integer I = new Integer(33);

i.intValue()

Static Integer valueOf(String s)

//Convert string to wrapper class

Static methods are invoked by

Classname.MethodName() // Integer.parseInt

Byte wrapper class

To perform operation on the byte datatype

Constructor

Byte(byte c)

{

}

Byte b1 = new Byte((byte)14);

Short s1 = new Short((short)20);

Long l = new Long(10);

Boolean

To pesorm the operation on Boolean datatype

Boolean( Boolean b1)

{

}

Boolean( String a)

{

}

Float wrapper class

Constructor

Float(double d)

Float f1 = new Float(3.14);

Float(float f)

Float f2 = new Float(3.14f);

Float(String s)

Float f3 = new Float (“3.14”);

Methods

Static int compare(float f1, float f2)

//compare the float values

Float.compare(a,b)

Int compareTo (Float f)

// compare float object

Float f1 = new Float(3.14);

Float f2 = new Float(3.14);

Sop(f1.compareTo(f2));

Static Boolean isNan(float f); // check datatyte

Boolean isNaN() // object

**Double wrapper class**

**Constructor**

Double which takes double as an argument. Double which takes string as an argument

Double( double d1)

{

}

**Character wrapper class**

It takes only character as an argument.

Character (char ch)

{

}

Static Boolean isLetter( char ch)

Static Boolean isDigit(char ch)

Static Boolean isLetterOrDigit (char ch)

Static Boolean isUpperCase(char ch)

Static Boolean isLowerCase(char ch)

Static Boolean isWhiteSpace(char ch)

**Autoboxing and Unboxing**

Available only from jdk 1.5

Automatic conversion of datatype to wrapper class(auto boxing) and wrapper class to datatype(unboxing)

With auto boxing

Integer i;

Int j;

I=5; // data type to wrapper class

J=10;

J=i ; // wrapper class to datatype

Without autoboxing

Integer i ;

Int j;

I=new Integer(5);

J=10;

J=i.intValue();

This keyword:

Refers to the current class instance variable.

Used when parameter names and instance variable names are same

This constructor

It is used to invoke different constructor of the same class

Always it should be present in the first line

Polymorphism

One func that takes many forms

1. Static or compile time polymorphism

Method overloading:

Same method name but different number, order and datatype of the argument present in the same class.

**Varargs**

Variable number of arguments. Passing variable number of arguments to a method and available from jdk 1.5 version

Representation (…)

Method can have only one Variable argument that is present at last.

Method can have only one Varargs

**Non access specifiers**

It has its own restriction where as access specifiers is applied for every thing

1. static

It is a non access specifier

When a class is declared to be static , no need to create an object.

Only inner classes can be static, outer class should not be static.

When a method is declared to be static , it is invoked by using className.methodName(); condition if it is present n different class.

Same class 🡪 method name()

If variable is declared static it is initialized only once and invoked as variable name in same class

Different class -- > classname.variable\_name

Static method can access only static content.

Otherwise, we have to create an object and access.

If a class contains a static block, then static block will be executed before the main method

System -- >class

Out -- > object of PrintStream (static field)

Println 🡪method of PrintStream

Static import

Available from jdk 1.5 where we can call static methods and static variables directly without using class name.

**import** **static** java.lang.Math.\*;

**import** **static** java.lang.System.\*;

**public** **class** Main10 {

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

// **TODO** Auto-generated method stub

**double** d =*sqrt*(16);

***out***.println(d);

}

}

Inheritance

Super keyword

Only in base class constructor super() should present in first line.

How constructors are invoked in inheritance

In inheritance , the constructor always invoked as top down approach. First it goes to the related class constructor and checked we used this or super. Otherwise it will try to invoke only the default constructor of the base class.

{} --🡪 Compound block or instance block ( Invoked before constructor invokes)

**Is a relationship and Has a relationship**

Where ever inheritance comes , then that of relationship is called as is a relationship.

Trying to create a object of one class into another class, is called as has a relationship

**Dynamic polymorphism**

(method overriding)

Same method name, same return type of the method, same number order and datatype of the argument present in different class and the class should be inherited.

In order to have effective method overriding , we have to adopt dynamic method dispatch..

Here we create a object of base class and store the reference of derived class.

At compilation , the compiler thinks object is created for base class , only at the time of execution it knows the object contains the reference of derived class. In this way we achieve run time polymorphism.

**Final keyword**

It is a non access specifier

If a class is declared as final, it cannot be inherited.

If a method is declared as final, it cannot be overridden.

If a variable is declared as final, it cannot be changed.

**Accessor and mutator methods (getter and setter method)**

**Object class**

It is the super class of all class.

**Constructor**

Object()

{

}

**Methods**

String toString()

Boolean equals(Object o)

int hashCode(); // returns the internal address for the value stored in the object

If two object are equal according to equals method , hashCode method produces same integer.

Protected void finalize() //discussed in Garbage collection

Final void wait()

Final void wait(long millisecond)

Final void notify()

Final void notifyAll()

**Abstract keyword**

It is a non access specifier

When a class is declared to be abstract it cannot be instantiated.

When a method is declared to abstract it doesnot contain definition , it end with semi colon.

Variables cant be abstract.

A class should be declared as abstract if it contains an abstract method but not necessarily all the abstract class should have abstract method.

The abstract class can also contains some normal methods.

The abstract class can also be inherited, at that time we have to compulsorily give the definition of the abstract method in the inherited class or define the class itself to be abstract.

Abstract class contains a default constructor.

**Interface**

Instead of mutilple inheritance we use interface.

It contains method declaration and variable declaration and initialization.

Interface are syntactically similar to class, it contains method declaration and variable declaration and initialization

Syntax:

Access\_specifier Interface Interface\_name{

//method declaration

//variable declaration and initialization

}

By default all interfaces are abstract, so we cant create an object for interface

By default all interface methods are public and abstract.

By default all interface variables are public, static , final.

So interface variables are accessed using InterfaceName.Variable\_name

Interface are using implements keyword.

If a implements an interface, we have to compulsorily provide the definition of interface methods with public access specifier or make the class itself to be abstract.

The implemented class can aslo contain some normal methods.

The interfaces can also be inherited.

In java , we have one class extends one class.

One class implements many interface

One interface extends many interface.

An interface without any methods and variables are called as marker interface

Example

Serializable ,clonable , remote etc

**Exception handling in java**

Error : That are generated at compile time

Exception: Generated at run time

It is of two types : check exception and unchecked exception

Checked exception : All the subclasses of the exception is checked exception. It will insist the programmer to surround the code using try catch or throws otherwise the compiler will not compile the program.

Unchecked exception : All the subclasses of runtime exception is unchecked exception. The compiler will not insist the programmer to surround the code using try catch or throws . But at the time of execution it throws the exception.

1.Arithmetic exception:

Divide by zero

2.Array Index out of bounds exception:

3.String index out of bounds exception:

4.Array store exception

Storing invalid elements in the array.

5.Negative array size exception

Try to declare the array with negative size.

6.class cast exception:

Store the reference to derived class into base class, if it is done wrongly then it is class cast exception

7.Null pointer exception:

Invoke anything with the help of null reference

8.Number format exception:

Converting “abc” to integer

9.Illegal monitor state exception.

10.Illegal thread state exception

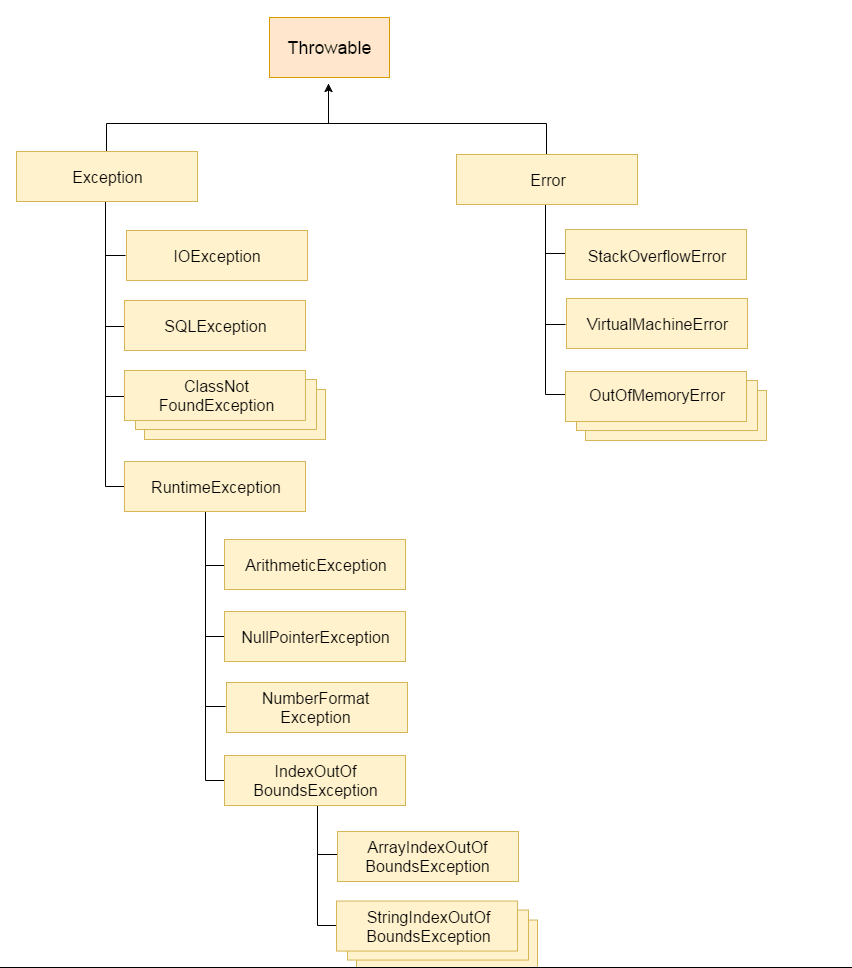
How to handle exception

Keywords to handle exception.

Try – program to be monitored for exception is put inside try block.

Catch- used to caught the exception.

Finally- it is an optional statement used to close the resources, if it is used whether exception occurs or not it will be executed after the try block. Mainly used in file programing , database programing ,socket programing.



Multi catch statement

A single try can have multiple catches.

Jdk 1.7 , there is a new concept for multi catch block.

**try**

{

**int** b=42/0;

**int** [] a = {1};

a[33]=42;

}

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

{

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

}

General class exception is not possible here.

Throw keyword:

It is used to manually throw an exception.

Syntax:

throw new ArithmeticException()

{

}

Throws keyword:

Used to throw the exception, and executes as usual it can be used only in methods

**User defined exception or custom exception**

**Package : java.util.\*; (utility package or collection framework)**

Collection frame work is used to store the collection of objects.

It is provided with two things.

1.Collection Interface

2.Collection class

It is a core interface to store a collection of objects.

Collections class:

It is provide with static algorithm to support util package.

Collection interface

Sortedset

List Interface Set Interface

treeset

ArrayList,LinkedList,Vector Hashset,Linked

Hash set

Ordered,duplicates allowed unordered, no duplicates

1.ArrayList class

It is a dynamic array used to increase or decrease its size at run

Time.

Used for faster selection and slower insertion , deletion

Default capacity is : 10

Constructors:

1.default constructors

2.constructoe(int capacity)

3.constructor(Collection c)

Methods

Void add(Object obj)

{

// to add single object

}

Void add (int index, Object obj)

{

//add single element at particular position

}

Void addAll(Collection c)

{

//add collection of object

}

Void addAll(int index,Collection c)

{

}

Object get(int index)

{

}

Void remove(int index)

{

}

Object set(int index,Object obj)

{

}

Int indexOf(Object obj)

Int lastIndexOf(Object obj)

Boolean contains(Object obj)

Int size()

Boolean isEmpty()

Generics concept:

List <Employee> l = new ArrayList<Employee>(); //generics from jdk1.5

List <Employee> l1 =new ArrayList<>(); jdk1.7

**LinkedList**

Similar to ArrayList .

It is used for faster insertion and deletion and slower selection.

It implements List,Deque( where insertion and deletion at both the ends)

**Vector**

Similar to ArrayList but it is synchronized or thread safe.

It is a legacy class

Default capacity is 10

Constructor

Vector()

{

}

Vector(int capacity)

{

}

Vector(int capacity,int increment)

{

}

Vector(Collection c)

{

}

**Set Interface**

**Hashset**

It is used to print the element in the random order.

Default capacity : 16

Whenever we add the elements into hash set, internally it

Creates map set.

Set<String> s =new HashSet<>();

s.add(“A”); m.put(key,value); m.put(“A”,dummy object);

s.add(“B”); m.put(key,value); m.put(“A”,dummy object);

Whenever we add elements into hashset internally it creates a

map which contains inserted value as a key and dummy object for the value.

Constructors:

HashSet()

{

}

HashSet(int capacity)

{

}

HashSet(int capacity,float fillratio)

{

}

HashSet(Collection c)

{

}

Fill ratio : -

It ranges from . Default 0.75. If the capacity is lesst han the number of elements , it is multiplied by fill ratio in order to increase the capacity.

Ex : (3\*0.75) + 3;

LinkedHashSet:

Prints the element in the same order.

Treeset class

It is used to sort the elements in alphabetical or ascending order.

Map interface

It is used to store collection of object as a unique key value pair.

Map is unordered

Map.Entry interface

It is used to access the key and values separately

Insert element into map interface.

Put() -- > method

Map interface method :

Void put (Object key, Object value)

Void putAll(Map m)

Object get(Object key)

Void remove(Object key)

Boolean containsKey(Object key)

Boolean conatinsValue(Object value)

Set entrySet() – converts both key and values to Set interface.

Set keyset() – converts only key to into Set interface.

Map interface --------implements----------------------- Abstract

Map

Hash Map

Hash Map:

Contain unique key valur pair and prints in random order.

It is a class.

Constructors : It contains constructors as of hash map

LinkedHash Map:

It contains key value pairs and print in the order which we have inserted.

TreeMap

It is used to sort the elements based on the keys.

HashTable

It is a legacy class.

It is similar to HashMap but it is synchronized or thread safe.

Properties class:

It is a sub class of hash table . It print in unordered fashion . Here both key and value must be String.

Date class:

It is used to print date and time

Constructor:

Date()

{

//print current date and time

}

Date(long millisecond)

{

// prints the date and time from Jan 1st 1970

}

Methods:

1. int compare(Date d)

{

//compare two date

}

2.Boolean before(Date d) , Boolean after(Date d)

3.void setTime(long millisecond)

4.long getTime()

Calendar

It is a abstract class used to extract the useful information from date and time component.

Constructor:

Calendar()

{

}

Methods:

Public static Calendar getInstance() // create object for class

Public int get( int field)

Public int add(int field,int value)

Public int set(int field,int value) // replace

Public Boolean before(Calendar c)

Public boolen after(Calendar c)

Public int getActualMaximum(int field)

Public int getActualMinimum(int field)

Public int getMaximum(int field)

Public int getMinimum(int field)

Final Date getTime(); //converts Calendar to Date

Final void setTime(Date d)://converts date to calendar

**Gregoriancalendar**

It is concrete implementation of calendar class , where we can create objects. It is used to extract information from date time components.

Gregoriancalendar()

{

}

// It prints current date and time.

GregorianCalender( int year,int month,int date)

{

//my own date

}

GregorianCalendar(int year,int month,int date,int hours,int minutes,int seconds)

{

}

Methods:

Methods as of calendar

Boolean isLeapyear(int year)

Constants

Public static final int YEAR ;

Public static final int MONTH ; (STARTS WITH 0)

Public static final int DATE;

Public static final int HOUR;

Public static final int MINUTE;

Public static final int SECOND;

Public static final int MILLISECOND;

Public static final int ERA;

Public static final int AM/PM;

Public static final int MONDAY TO SUNDAY;

Public static final int JANUARY TO DECEMBER

Public static final int DAY\_OF\_MONTH;

Public static final int DAY\_OF\_WEEK;

Public static final int DAY\_OF\_YEAR;

Public static final int WEEK\_OF\_YEAR;

Public static final int WEEK\_OF\_MONTH;

Public static final int HOUR\_OF\_DAY;

SimpleDateformat class--- > It is used for formatting (Date to String) and parsing (String to date)

SimpleDateformat()

{

}

Constructor which takes string pattern

SimpleDateFormat(String pattern)

{

}

Different types of pattern

G 🡪 Represents the Era.

y🡪 Represents year in number

Y 🡪 Represents the week of year

M 🡪 Month in year

W 🡪 Week in month

D 🡪 Day in year

d-🡪 Day in month

E🡪 Day name in week

a🡪 AM/PM

H🡪 Hour in day (0 to 23)

k🡪 Hour in day ( 1 to 24)

K🡪 Hour in AM/PM (0 to 11)

h🡪 Hour in AM/PM (1 to 12)

m🡪 Minutes

s🡪 Seconds

S 🡪 miiliseconds

Methods:

Void setLenian (Boolean on)

Date parse (String s)

String format(Date d)

**Java.io.\*;**

It is provided with two classes 1.file class and 2.Stream class

1.File class

Access info about existing file

2.Stream class

Perform operation on file

2a . ByteStream (read and write in bytes)

2b.CharacterStream

(read and write in char)

Bytestream is divided into two class

Abstract Inputstream(Read in bytes) and Abstract OutputStream(Write in bytes)

FileInputStream and FileOutputStream.

CharacterStream

Abstract reader and Abstract Writer

FileReader and FileWriter

Reader class

It is used to read in the form of characters.

Int read //used to read a single character

//returns -1 if end of file is encountered.

Int read(char [ ] c)

//used to read a array of characters and returns the number of bytes read

Int read(char[] c, int offset,int numchars)

{

}

FileReader

It extends Reader

It is used to read data from file in the form of characters.

If the file does not exist , it throws FileNotfoundException

Constructor

FileReader(String FilePath)

FileReader(String DirectoryPath,String FileName)

FileReader(File f)

Writer class

Abstrcat class used to write in the form of characters.

Methods:

Void write(int v )

Void write(char[] c)

Void write(char[]c,int offset,int numchars);

Void close()

Void flush()

FileWriter class extends writer.

It is used to write in a file , in the form of characters

If the file doesnot exist, it creates a new file. If exists it overrides the content.

FileWriter(String FilePath)

FileWriter(String FilePath,Boolean append)

**String Buffer**

It is present in java.lang.\*;

It is a mutable class with variable number of characters.

It is synchronized or thread safe and we cant override the equals method in string buffer

**Constructor**

**1.Default constructor**

2.StringBuffer(String s)

3.StringBuffer(int capacity)

Default capacity : 16

Int length()

Void setCharAt(int loc,char c)

Void setLength(int len)

char charAt(int loc)

void append(String s)

void append(int a)

void append(Object o)

void insert(int I,String s)

void reverse(String s)

void delete(int start,int end)

void deletecharAt(int loc)

void replace

(int start , int end,String s)

**String Builder class**

It is similar to String Buffer . But it is not synchronized and so it gives a better performance.

**Threading**

It is used to run multiple lines of code in a single program simultaneously using thread class.

Syntax:

Public class Thread extends Object implements Runnable

{

}

Constructors:

Thread()

{

}

Thread(String thread\_name)

{

}

Thread(Runnable r)

{

}

Thread(String thread\_name , runnable r)

{

}

Thread(Thread\_group tg,String thread\_name)

{

}

Thread(ThreadGroup tg, Runnable r, String thread\_name)

{

}

Three constants

ThreadPriority.min = 1;

ThreadPriority.max=10;

ThreadPriority.norm=5;

Methods:

Public void start()

{

}

Public static void sleep(long miilisecond)

{

}

Two types of thread:

Main Thread and child Thread

Main Thread

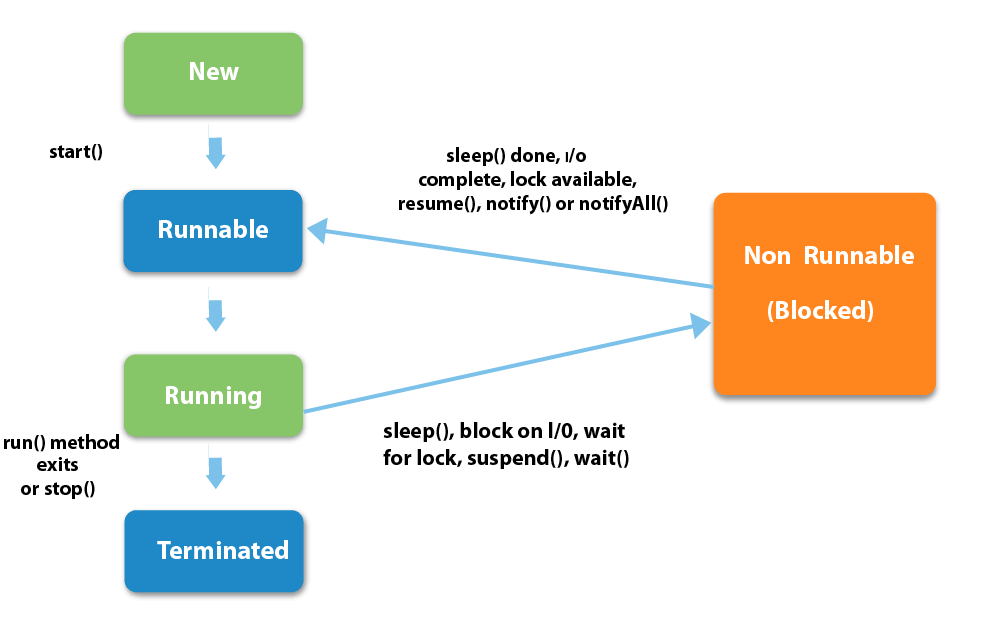
Where ever PSVM present by default a thread will be running called main thread.

It will be executed first and blocked to invoke the child thread.

Child Thread

We create ‘n’ number of child thread.

Life cycle of child thread



**Creation of child thread:**

Two ways to create a child thread.

By implementing runnable interface.

It acts like a child thread, to create the thread we use any one of 7 constructors.

Runnable interface contains one method,

Public void run()

{

}

//which contains the operation or the processing of child thread

By extending thread class

Extends Thread

Synchronisation : When multiple thread access a single resource , the thread hast to utilize the resource completely or fully is call synchronization. It can be achieved using synchronized keyword.

It is a non access specifier. It can be applied only to methods or synchronized block of objects.

Synchronized(object)

{

}

**Annotation:**

Instead of writing xml file we can use annotation.

Available from Jdk 1.5

Annotations are like meta data, means you are free to add your code and also apply htem to variable parameters,methods , constructors and class. Annotations are used instead of xml file.

Annotation contains two things

1.Annotation type

Creation of annotation

Syntax

Public @interface annotation\_name{

}

Rules to followed:

Method declaration should not throw any exceptions.

Method declaration should not contain any parameter.

Method declaration should have a return type..

Either string , datatype or object

2.annotation

Applying the created annotation in the java program either class level or method level

Two types of annotation:

1.Simple annotation

Present in java.lang.\*; package and it can be applied only on the java programs

Three types

[1.@override](mailto:1.@override)

Indicates the same type of method present in base class

Ensures that the annotated method is used to override the method in the super class.

[2.@deprecated](mailto:2.@deprecated)

This type of annotation ensures that the compiler warns you when you are using the deprecated elements of the program.

[3.@supresswarning](mailto:3.@supresswarning)

Compiler will shield the warning message in the annotated element

2.Meta annotation

Annotation about annotation and used at the time of creating a new annotation.

Present in java.lang.Annotation.\*; package

[1.@target](mailto:1.@target)

Specifies where the annotation is applied in the java program.

Like @target(Element Type.METHOD)

@target(ElementType.CONSTRUCTOR)

@target(ElementType.VARIABLE)

@target(ElementType.TYPE);

[2.@retension](mailto:2.@retension)

Specifies where and how long annotations are to be return.

@retension(RetensionPolicy.Class)

At the compile time

@retension(RetensionPolicy.RunTime)

[3.@document](mailto:3.@document)

It is used to document all methods,constructor of a class,variable in a html file.

Javadoc Annot\_ex.java -d doc

Javadoc filename.java –d doc

[4.@documented](mailto:4.@documented)

It is used to document the annotation also using Javadoc tool.

**JUnit**

It is a framework.

It is a unit testing open source framework for java programming language. It is used to test individual classes.

Configure the Junit jar file inside the project.

Org.Junit.Assert class provides set of assertion methods useful for writing test.

Void Assert Equals(Boolean expected , Boolean actual)

{

}

Void Assert False(Boolean condition)

{

}

Void AssertNotNull(Object obj)

Void AssertNull(Object ibj)

Void AssertTrue(Boolean condition)

Junit annotation:

@test

This a test method to run.

@before

Run before at test

@after

Run after @ test

@beforeclass

Run once before any of the test method

@afterclass

Run once after any of the test method