14/4/2024

Wrapper Class

First OOP ⇒ Small talk

Java = Not Pure OOP because of Primitive datatype
↓
{used because of performance reasons}

Primitive datatype	Wrapper class
byte	Byte
short	Short
int	Int
char	Character
long	Long
float	Float
double	Double
boolean	Boolean

{Super class of all numeric datatype is "Number"}

In JDK-5 we got, Autoboxing & unboxing

Before JDK-5 for integer we have to do:→

Integer i1 = new Integer (10);

Same

from JDK-5 onwards

int i = 5

Integer i1 = i;

Autoboxing

Integer i1 = 20; } Autoboxing

int i2 = i1; } Auto Unboxing

Boxing ⇒ (Converting) Assigning primitive to object reference of wrapper class.

Unboxing ⇒ (Converting) Assigning of wrapper class reference to Primitive

Integer i1 = 5; } boxing

Integer i2 = 10; } boxing

Integer i3 = i1 + i2

i1 → Unboxing → Int; i2 → Unboxing → Int; Int + Int → Int → Boxing → i3

```
Class A {
    PSVM(String [] args){
        int a = Integer.parseInt(arg[0]);
        double d = Double.parseDouble (args[1]);

    }
}
```

→ java A 10 20.5

{String is not a wrapper class}

Object

→ All classes in java inherits features of Object Class.

→ Some methods ⇒

① int hashCode () ← hashcode is a unique code of each Object.

⇒ Consistency during execution →

[in a single execution, hashcode of an Object will be same everytime]

⇒ Object value equality implies hashvalue equality →

```
A a1 = new A();
A a2 = a1;
```

[hashcode of a1 = hashcode of a2]

⇒ Object value inequality places no restriction on hash value

It is recommended that two object's hash code must be different, but not necessary, means it could be same as well, which is called hashing Collision {where hash value of multiple objects are same}

② equals() :→

→ To check equality of two objects.

→ By default it checks equality of object references.

→ For content wise checking, we must override the equals methods. Some classes like string have overridden this method as well.

→ hashcode () & equals () methods of objects are interrelated, if we override one of the method, then others one should be overridden.

```
Class A {
    int a, b;
}
```

```
A a1 = new A()
A a2 = a1;
a1.a = 5 , a1.b = 10;
A a3 = new A();
a3.a = 5 ; a3.b = 10;
```

```
if (a1 == a2){ // True {Same values}
    SoP("value");
}

if(a1 == a3){
    SoP("value2");
}
```

```
if( a1. equals (a2)) // True {Same Object}
{
    SoP("Object");
}
if( a1. equals (a3) )// false
{
    SoP("different Object");
}
```

Output ⇒ Object ⟸

If we want to compare content/values then we have to override equals() in A class

```
class A {

    int a, b;

    P. boolean equals (Object o){

        A ax = (A) o;
        if (ax.a == a && ax.b == b)
            {
            return true;
            }else{
                return false;
            }
    }
}
```

```
if (a1.equals(a2))
if (a1.equals(a3))
if (a1 == a2)
if (a1 == a3)
```

True

(3) toString() :→

```
class A {

    int a, b;
```

```
A a1 = new A();
a1.a = 5;   a1.b = 20;

SOP(a1);
```

internally

```
SOP( a1. toString());
```

(Classname @ Hashcode)

```
String s1 = "Hello" + a1;
```

a1. toString();

Hello A@123ab...

A @ 123 ...

L A; @ 123 ...

Reference Type — Class Name

(4) finalize() {depriciated}

→ works as destructor

→ Simple method which gets called automatically just before Object's execution ends.

String Class

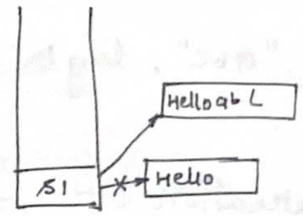
→ Java Strings are Objects

→ String is a final Class.

→ Immutable

Cannot change specific object (String), if we change it will create new object

```
String s1 = "Hello";
s1 = s1 + "abc";
```



Construtors of String →

→ String()

→ String(String str)

→ String(Char [] ch)

→ String(Char [] ch, int offset, int len)

Example ⇒

① String st = new String();

② String st1 = "abc";

③ String st2 = new String(st1);

④ Char ch = {'a', 'b', 'c', 'd'}

String st3 = new String (ch);
↓
abcd

⑤ String st4 = new String (ch, 2, 2);
↓
cd

Byte [] b = {1, 2, 3}

⑥ String st5 = new String (b)
↓
will pass ascii values

{ String literals are Objects }

sop("abc". length()); ⇒ (3)

Concatenation (+) ⇒ "Hello" + "abc";

Hello abc

⇒ "Hello" + 2 + 2; ⇒

Hello 22

⇒ 2 + 2 + "Hello";

4 Hello

Character Extraction Methods

Char ch = "abcd". charAt(2);

// c

⇒ Char c[] = "abcd". toCharArray();

String s1 = "Hello";

Char [] c = s1. toCharArray();

byte b[] = "abc". getBytes();
↓
(array of ascii code of each values)

16/4/2024

To check equality ⇒

① equals() { Overridden in String to check Contents }

② equalsIgnoreCase()

"Hello". equals ("hello"); // False

"Hello". equalsIgnoreCase ("hello"); // True

③ "Hello". CompareTo ("hello"); return < 0

↓
Integer value { difference of Ascii value }

{ Whenever it gets non-zero diff. it will return value }

H	e	l	l	o
↕	↕	↕	↕	↕
h	e	l	l	o

{ as this is different, so it will return Ascii of 'H' − Ascii of 'h' }

④ startsWith()

endsWith()

"HelloWorld".startsWith("Hello"); // True

"HelloWorld".endWith("World"); // True

"HelloWorld".startsWith("lo", 3); // True

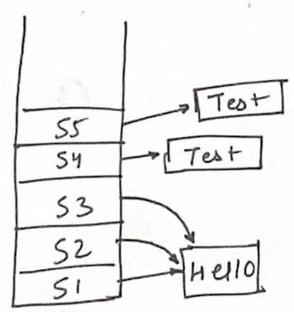
⑤ substring

"Hello World".substring(5); // World

"Hello World".substring(2, 4) ⇒ ll

In case of String Memory allocation will be done based on value that's why S1 & S2 with same value refers to same object

```
String s1 = "Hello";
String s2 = "Hello";
String s3 = s1;
String s4 = new String("Test");
String s5 = new String("Test");
```

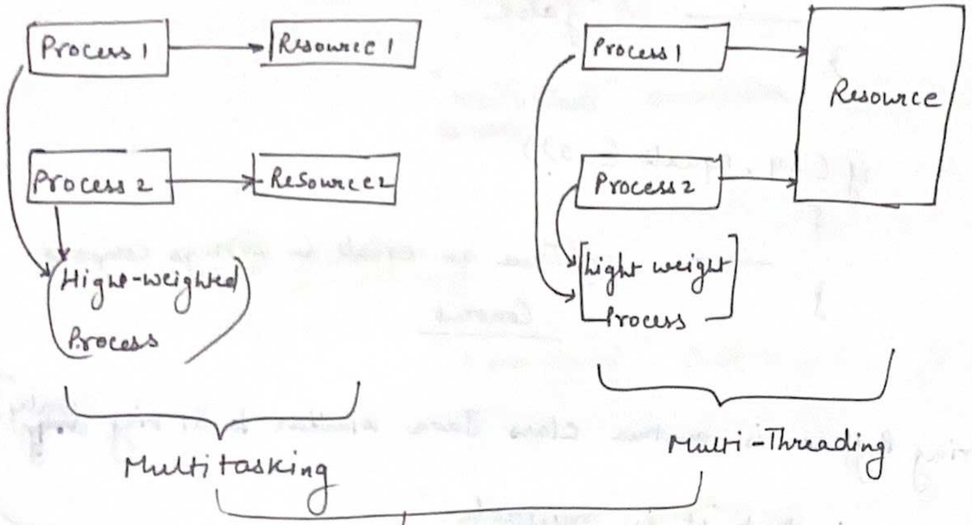



```
if (s1 == s2)
{
    ____        // True as s1 & s2 are referring to
}                  same object

if (s4 == s5)
{
    ____        // false
}

if (s4.equals(s5))
{
    ____        // True as equals in strings compares
}                  Content
```

String Buffers is another class Java similar to String Only difference is that it is mutable

Multithreading

Thread ⇒ - Single sequence of statements in execution.
- Light-weight processes



Only one process will execute at one time, because generally we have only single processors, Switching between these processes is really fast. thats why it seems like processes are executing simultaneously which they are not

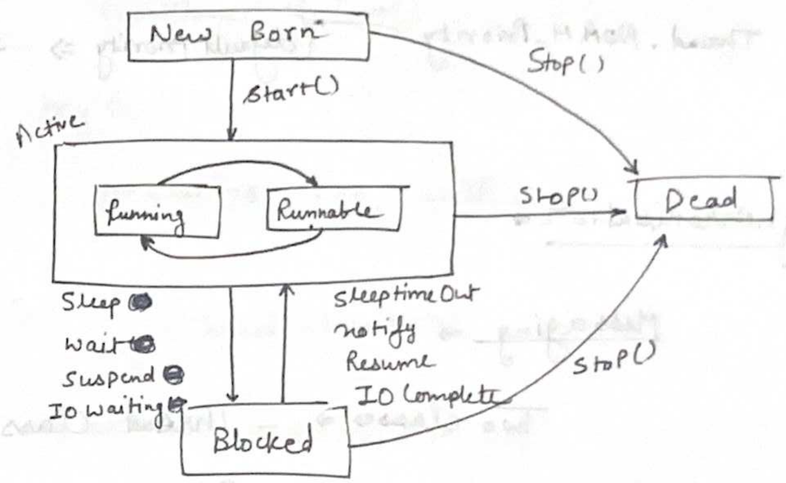
When we use multiple processors to execute multiple processes simultaneously that is known as Multiprocessessing.

Multi-Programming ⇒

Multiple-Programs
- Multi-tasking
- Multi-threading

⇒ did C/C++ Support Multithreading ⇒ Yes, but problem with C & C++ is, for Multithreading to Manage resources & threads we require "Scheduler" and in case of C/C++ it is on programmer to write their own "Scheduler" where as in Java, Java has its own Scheduler.

When we Create thread ⇒ It goes through multiple stages:→



16/4/2024

→ Scheduling is Round Robin + Preemptive but this is dependent on O.S.

Priorities ⇒ Thread.Max_Priority (10) to Thread.Min_Priority (1)

(10) — Highest ~~Lowest~~ ; (1) — Lowest

Thread.NORM_Priority → {default priority ⇒ 5}

Synchronisation ⇒

Messaging ⇒

Two classes ⇒ – Thread Class
– Runnable Interface

Thread ⇒

- get Name()
- get Priority()
- set Name()
- set Priority()
- join
- run
- isAlive
- Start
- sleep
- yield
- Current Thread()

```
Class MyThread {
    PSVM(_){
        Thread t = Thread.CurrentThread();
        SoP("Name:" + t.getName());
        SoP("Priority: " + getPriority());
        t.setPriority(10);
        t.setName("My Thread");
        try {
            for (int i=5 ; i>0; i--){
                SoP(i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e){
        }
    }
}
```

```
Class MyThread extends Thread {
    PV run() {
        try { int i;
              for(i=0; i<5; i++) { SoP("My Thread:" + i);
                                   Thread.sleep(1000);
        } catch (InterruptedException e){
        }
```

```
class ThreadDemo{
    PSVM(_){
        MyThread m1 = new MyThread();
        m1.start();
        try{
            for(int i=0; i<5; i++){
                SOP("Thread Demo: " + i);
                Thread.sleep(1000);
            }
        catch(InterruptedException e){

        }
    }
}
```

same as previous but with minor change

```
class Mythread implements Runnable{
    PV run(){
        =
        =
        try{
        -
        -
        } catch(_)
        -
        }
}
```

```
class ThreadDemo{
    PSVM(_){
        Mythread m1 = new Mythread();
        Thread t = new Thread(m1);
        t.start();
        //same try-catch as previous.
    }
}
```

As Runnable Interface doesn't have run() that's why we passed 'm1' to thread's object, as it expect 'Runnable' we can pass its subclass

17/4/2024

Two ways to create own thread

→ By extending Thread Class

→ By implementing Runnable Interface

Marker Interface {Related to Interface topic}

- Interfaces which don't have any method/member.
- are used to mark some specific feature for JVM.

ex ⇒ Serillizable, Clonable } Marker Interface

• isAlive() & Join() ⇒

- isAlive() → To check whether the method is in dead state or not
- join() ⇒ To wait for completion of some other thread.

```
main(){
    m1.join()
    m2.join()
    ⋮
}
```

Here main thread will wait till m1 & m2 goes to dead state.

Thread that is calling only that thread will wait for eg in ours example main is calling thread only main will wait till execution of all called thread with join.

```
Class MyThread extends Thread{
    PV run(){
        try{
            for(int i=0; i<10; i++){
                SOP(i);
                Thread.sleep(1000); }
        } catch(___){
        }
    }
}
```

```
Class A{
    PSVM(__){
        MyThread mt = new MyThread();
        mt.start();
        try{
            mt.join();
            SOP("After Thread");
        } catch(___){
            ___
        }
    }
}
```

This will print in the end

```
Class Mythread extends Thread{
    String nm;
    My Thread (String name){
        nm = name;
    }
    PV run(){
        try{
            for(int i=0; i<10; i++){
                SOP(nm + i);
                Thread.sleep(1000);
            }
        }catch(__){
            ___
        }
    }
}
```

```
Class A{
    PSVM(_){
        My Thread mt = new MyThread("One");
        My Thread mt2 = new MyThread("Two");
        mt.start();
        mt2.start();
        try{
            mt.join();
            mt2.join();
            SOP("After Thread");
        }catch(__){
            ___
        }
    }
}
```

Priorities ⇒

Set Priority	1 ——— 5 ——— 10
getPriority	

1 → Thread.MIN_PRIORITY
5 → Thread.NORM_PRIORITY
10 → Thread.MAX_PRIORITY

Synchronised ⇒ The block / method which can be executed by a single thread at a time.

→ Synchronised can be used with →

1. method
2. Block with Object.

ex ⇒

```
Synchronised void m1(){
    ___
}
```

Only one thread can execute m1() at a time.

```
A a1 = new A();
Syncronised (a1){
    ___
}
```

In this block, all calls made to a1 will be synchronised.

Suspend, Resume, Stop ⇒ → Depricated

New Versions

Wait, notify(), notifyAll()

18/4/2024

~~Collections~~ Generics

→ Datatype Independent Classes.

→ Datatype can be defined at the time of creation of object.

→ eg ⇒

```
Class Stack{
    Integer i;
}

Class Stack{
    String s1;
}

Class Stack{
    Student s;
}

Class Stack <T>{
    T i;
}
```

If we don't pass any <T> while creation, T will be Object by default.

```
Stack<Integer> S1
    = new Stack<Integer>();

Stack <String> S2 = new
    Stack<String>();

S1 = S2;
S2 = S1;
```

different

```
Class A <T1, T2>{
    T1 a;
    T2 b;
    :
}
```

→ `A <Integer, String> a1 = new A <Integer, String> ();`

If we want that T can be only be numeric

```
Class A <T extends Number>{
    T a;
    :
}
```

Collection Framework

- was added in JDK-2
- we have things like LinkedList, Stack, Hashtable, but why we need these,
  - High performance
  - Common framework.
  - Extendability.
- In the framework we got,
  - Set of Interfaces.
  - All methods of legacy classes are synchronised.

whereas method of collection framework are non-synchronised.

- High Performance and
- Huge set of APIs

(Collection of elements, no sequence)

Collection

- List (sequence of elements)
- Set (Set Properties) (No duplicacy, No sequence)
  - SortedSet (Set properties, with sort property)

Some Interfaces ⇒

(would work on any collection)

① **Iterator** ⇒ To iterate the elements of Collection.

- hasNext() - next elements exists or not.
- next() - get next element.
- remove() - to remove current element

(Sub-Interface of Iterator) (would only work on Lists)

② **ListIterator** →

- add(Object)
- hasNext()
- hasPrevious()
- next()
- previous()
- nextIndex()
- PreviousIndex()
- remove()
- set(Object)

③ ⇒ **RandomAccess Interface** ⇒ {Marker Interface}

- To mark that it will Access elements in Random

④ ⇒ **Comparator** :→

int compare (Object ob1, Object ob2)

Collection Interface ⇒

- add(Object o)
- addAll(Collection c)
- Clear() — remove all elements of collection.
- Contains(Object o) - Check whether o is available in Collection.
- ContainsAll(Collection c) - check whether present collection have all elements of c.
- equals(Object)
- isEmpty()
- iterator()
- remove(Object)
- removeAll(Collection)
- retainAll(Collection c) → retain only 'c' and remove all.
- size()
- toArray()
- toString()