**Constructor:**

Main task for constructor is to provide initialization for newly created object.

**Q) class** Test2{

**int** x=10;

{

System.*out*.println("parent SB");

}

Test2(){

System.*out*.println("Parent Constructor");

}

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

Test2 obj= **new** Test2();

System.*out*.println("Parent main method");

}

**public** **void** m1(){

System.*out*.println(y);

}

**int** y=20;

}

**class** Child **extends** Test2{

**int** i=100;

{

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

}

Child(){

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

}

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

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

}

}

Java Child:o/p: child main

Instance blocks are only called when constructor is called. While creating object, if we want to perform initialization, go for constructors. Except for initialization, any other task should go in instance block. Constructors can take arguments, instance blocks cannot. If a class has more than 1 constructor, duplicate code can go in instance block.

The applicable modifiers for constructor are: public, private, protected, default. None other is allowed. The constructor access modifier is by-default same as that of the class. The first line of the constructor will be either super() or this(). By default, compiler places super() as the first line. Super() and this(), both together cannot be used and call to super()/this() can only be the first statement in a constructor, should not be in a method. We cannot use this/super in a static method/block.

Singleton classes: Only 1 object for a class. Eg: Runtime.getRuntime() returns single instance. Private constructor and factory method has to be provided. Interfaces cannot have a constructor.

Q) class Test{

Public static void main(String args[]){

Sysout(super.hashcode());

}}

o/p CE: cannot use super/this in a static block/method.

Typecasting:

A b= (c) d;

Compiler rules:

1. C and d must have some relationship
2. C must be either derived type of A else Incompatible type error.
3. D must be either same type or derived type of C, else classCastException.

**Collections**

Collection: interface.

Array: fixed size, indexed collection, homogeneous data elements. By memory point of view, not a good choice, but performance wise, we must use arrays. Difficult to insert/ delete element in mid

**byte** a= 5;

**int** x=a;

Object[][] obj= **new** Object[20][];

obj[0]= **new** Integer[1];

obj[1]= **new** String[3];

Integer[] arrInt= **new** Integer[4];

arrInt[0]=5;

arrInt[1]= **new** Integer(30);

arrInt[2]= **new** Byte(4);//CE

arrInt[3]= a;//CE, cannot convert from byte to Integer

A11[] arr= **new** A11[2];

arr[0]= **new** B11();

arr[1]= **new** A11();

}

**class** A11{

}

**class** B11 **extends** A11{

}

Collections: class in java.util package.

List: insertion order is preserved. Duplicates are allowed. Eg: arraylist, linkedlist, vector, stack.

Set: duplicates not allowed. Insertion order not preserved. Eg: hashset, linkedhashset.

Sortedset: sorting order.

Navigableset: for navigation.

Queue: FIFO eg: priority blocking queue, linkedBlockingQueue, priority queue

Map: key value pair. Duplicate keys not allowed.

SortedMap: sorting based on keys.

Legacy classes: vector, stack, enumeration, dictionary, hashtable, properties.

Collection: add, addAll, remove, removeAll, retainAll, clear, isEmpty, size, contains(Object), containsAll(Collection), toArray, iterator.

ArrayList: Insertion order preserved. Null insertion is allowed. ArrayList and vector implements RamdomAccess interface. Not good for insertion/ deletion in middle as it requires shifting. ArrayList is not synchronized. Vector is synchronized. To synchronize arraylist, Collections.synchronizedList(l).

ArrayList list= new ArrayList();

By default 10 is the initial capacity. New capacity = 1.5\* initial capacity +1;

ArrayList list= new ArrayList(capacity);

ArrayList list= new ArrayList(collection);

Q) List list= **new** ArrayList();

list.add(5);

list.set(0,1.1f);

list.add(**null**);

System.*out*.println(list);

o/p: [1.1, null]

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

List list= **new** ArrayList();

list.add(5);

list.add(0,1.1f);

list.add(**null**);

System.*out*.println(list);

list.remove(2);

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

list.add(2, "error");

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

}

o/p: [1.1, 5, null]

2

3

Collections.*synchronizedCollection*(list);

Collections.*synchronizedList*(list);

LinkedList: Insertion order is preserved. Duplicated allowed. Null elements allowed. Implements Serializable, cloneable but nor randomAccess.

List list= new LinkedList();

List list= new LinkedList(Collection );

LinkedList generally used to implement stack, queue.

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

LinkedList list= **new** LinkedList();

list.add(5);

list.add(0,1.1f);

list.add(**null**);

System.*out*.println(list);

list.removeLast();

list.addFirst("hello");

System.*out*.println(list);

list.add(2, "error");

System.*out*.println(list);

Collections.*synchronizedCollection*(list);

}

o/p: [1.1, 5, null]

[hello, 1.1, 5]

[hello, 1.1, error, 5]

Vector: Vector vec= new Vector(); Initial capacity is 10. New capacity= 2\* initial capacity.

All methods are synchronized.

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

Vector list= **new** Vector();

list.add(5);

list.add(0,1.1f);

list.add(**null**);

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

System.*out*.println(list);

list.add(2, "error");

System.*out*.println(list);

Enumeration e= list.elements();

**while**(e.hasMoreElements()){

Sysout(e.nextElement());

}

}

o/p: 1.1

[1.1, 5, null]

[1.1, 5, error, null]

1.1

5

error

null

**Stack:** Only one constructor: search method returns offset from top of the stack. If not found, returns -1.

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

Stack a= **new** Stack<String>();

a.push("1");

a.push(3l);

a.push("hello");

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

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

System.*out*.println(a);

System.*out*.println(a.search("1"));

System.*out*.println(a.search("@"));

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

Enumeration e= a.elements();

**while**(e.hasMoreElements()){

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

}

}

o/p: hello

3

[1, 3]

2

-1

**Cursors: Enumeration: only to retrieve elements & only for legacy classes, Iterator: to retrieve and remove elements, ListIterator: to retrieve, remove, add elements.**

1. hasMoreElements()
2. nextElement()
3. Collection.elements()

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

Stack a= **new** Stack<String>();

a.push("1");

a.push(3l);

a.push("hello");

Iterator it= a.iterator();

it.remove();

**while**(it.hasNext()){

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

}

}

**o/p:** Exception in thread "main" java.lang.IllegalStateException

at java.util.Vector$Itr.remove(Unknown Source)

at Day2.Date.main(Date.java:17)

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

Stack a= **new** Stack<String>();

a.push("1");

a.push(3l);

a.push("hello");

Iterator it= a.iterator();

System.*out*.println(a);

**while**(it.hasNext()){

it.next();

it.remove();

}

System.*out*.println(a);

}

o/p: [1, 3, hello]

[]

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

Stack a= **new** Stack<String>();

a.push("1");

a.push(3l);

a.push("hello");

Iterator it= a.iterator();

**while**(it.hasNext()){

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

a.add(0, "4");

a.removeElementAt(0);

}

o/p: 1Exception in thread "main"

java.util.ConcurrentModificationException

**LinkedHashSet:** Insertion order is preserved.

SortedSet: Interface. Has methods like first(), last(), headset(): returns elements less than the object, tailSet(), subset(obj1, obj2): returns elements greater than or equal to obj1 but less than obj2, comparator();

Only way to retrieve objects out of a hashSet is by using iterator. While storing object in a hashSet, object is stored as a key and value is a constant. HashSet is implemented using hashmap.

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

SortedSet set= **new** TreeSet();

set.add("1");

set.add(2);

set.add(**null**);

set.add(3);

System.*out*.println(set);

}

o/p: Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer

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

Set set= **new** TreeSet();

set.add(2);

set.add(**null**);

set.add(3);

System.*out*.println(set);

}

o/p: Exception in thread "main" java.lang.NullPointerException

**TreeSet: ConcurrentModificationException:** Once iterator has been defined, if we modify the set other than the iterator remove method, may throw this exception. Not guaranteed in case of unsynchronized concurrent modification

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

SortedSet set= **new** TreeSet();

set.add("2");

set.add("4");

set.add("3");

Iterator it= set.iterator();

it.next();

set.remove("4");

System.*out*.println(set);

}

o/p: [2, 3]

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

Set set= **new** HashSet();

set.add(2);

set.add(**null**);

set.add(3);

System.*out*.println(set);

}

o/p: [null, 2, 3]

**TreeSet can have only first element as null. If we try and insert other elements, will throw NPE. HashSet can have null as an element. Heterogeneous elements are not allowed else classcastException.**

Q) System.out.println("hello".compareTo(null));

o/p: Exception in thread "main" java.lang.NullPointerException

Q) Integer x= 1, y=**null**;

System.*out*.println(x.compareTo(y));

o/p: NPE

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

Set set= **new** HashSet();

set.add(1);

set.add("hello");

set.add(**null**);

System.*out*.println(set);

}

O/P: [null, 1, hello]

For classes that follow natural sorting, it is ok to add to treeset without implementing comparable interface, else we have to. Otherwise classcastexception.

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

Set set= **new** TreeSet();

set.add(**new** StringBuffer("1"));

set.add(**new** StringBuffer("hello"));

set.add(**new** StringBuffer(**null**));

System.*out*.println(set);

}

o/p: Exception in thread "main" java.lang.ClassCastException: java.lang.StringBuffer cannot be cast to java.lang.Comparable

**Comparable Interface:** java.lang package. Has compareTo(obj) method. Returns –ve if less than obj, +ve if greater than and 0 if equals.

Q) **package** Day2;

**import** java.util.Set;

**import** java.util.TreeSet;

**import** Day1.Test2;

**public** **class** Date **extends** Test2{

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

Set<Student> set= **new** TreeSet<Student>();

set.add(**new** Student(1, "Rahul", "aggarwal"));

set.add(**new** Student(1, "Rahul", "aggarwal"));

System.*out*.println(set);

}

}

**class** Student **implements** Comparable<Student>{

**private** **int** rollNumber;

**private** String[] name= **new** String[2];

Student(**int** rollNumber, String... names){

**this**.rollNumber= rollNumber;

**int** i=0;

**while**(i<names.length ){

name[i]= names[i];

i++;

}

}

@Override

**public** **int** compareTo(Student s) {

**if**(**this**.rollNumber < s.rollNumber){

**return** -1;

}**else** **if**(**this**.rollNumber > s.rollNumber){

**return** 1;

}**else**{

**return** **this**.name.toString().compareTo(s.name.toString());

}

}

@Override

**public** String toString(){

**return** **this**.rollNumber+ " "+ **this**.name[0]+" "+ **this**.name[1];

}

}o/p: [1 Rahul aggarwal, 1 Rahul aggarwal]

**Treeset uses compareTo method for inserting elements in a set whereas hashset uses hashcode and equals method for this operation.**

**Comparator Interface:** compare(Object o1, Object o2),equals.

In case of Iterator and enumeration, we can only move in the forward direction. Using iterators, can only remove or retrieve elements.We cannot set or add element using an iterator. ListIterator is a child of Iterator interface. Can add, replace objects, can move in a forward or backward direction.

ListIterator: hasNext(), next(), nextIndex(), hasPrevious(), previous(), previousIndex(), remove(), set(Object), add(object)