Notes

1. Method without body is called abstract method.
2. Return 10; sends control back to calling method
3. Creating duplicate static variables is not allowed in same scope, its possible in different scopes . Ex.Duplicate field MEx.a
4. we can execute main method at the time of class loading by using static block.
5. main method is public why because it must be call from outside of package. Main method is mediator b/w jvm and java developer
6. If we modify static variable using one object that modification affected to all objects, because all objects share same copy of static variable’s memory location.
7. If we modify non static variable using one object that modification not affected to all objects, because all objects share own (separate) copy of non static variable’s memory location.
8. In the down casting we can get class cast exception , in java some scenarious allow us to perform downcasting.
9. A a=**new** B();// upcasting
11. // performing down casting implicitly we can get ce
12. //B b=new A();
13. // performing down casting Explicitly like this
15. B b=(B)a;// down casting

**Co-varient Return Type:**

We actually overring same method name same prototype and return type , but java 5 onwards introduced the concept co variant return type.

Covariant return type refers to return type of an overriding method. It allows to narrow down return type of an overridden method without any need to cast the type or check the return type. Covariant return type works only for non-primitive return types.

From Java 5 onwards, we can override a method by changing its return type only by abiding the condition that return type is a subclass of that of overridden method return type.

**publicclass** Super {

**public** Super get() {

System.***out***.println("super class");

**returnthis**;

}

}

**publicclass** Sub **extends** Super {

**public** Sub get() {

System.***out***.println(" Sub class ");

**returnthis**;

}

**publicclass** Test {

**publicstaticvoid** main(String[] args) {

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

s.get();

}

}

Op:Sub class

**Three important cases in method overloading**

1. When a method is overloaded with siblings parameters along with super and subclass parameters , if we pass null directly it leads ce: amboguos error

**publicclass** Super {

**publicstaticvoid** get(Object o) {

System.***out***.println("object method");

}

**publicstaticvoid** get(StringBuffer o) {

System.***out***.println("object method");

}

**publicstaticvoid** get(String o) {

System.***out***.println("object method");

}

**publicstaticvoid** main(String[] args) {

*get*(**null**);

}

}

1. If method is parameter is super and sub class type then we pass null value , then it gives importance to subclass.

**package** com.nare;

**publicclass** Super {

**publicvoid** m1(Super s) {

System.***out***.println(" Super class");

}

**publicvoid** m1(Sub s) {

System.***out***.println(" Sub class");

}

**publicstaticvoid** main(String[] args) {

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

s.m1(**null**);

}

}

**package** com.nare;

**publicclass** Sub **extends** Super {

}

Op: Sub class

3) When a method is overloaded with siblings parameters , if passed argument is matched with both parameters , if we pass null directly it leads ce: ambiguos error

Jogged array:

Multidimensional array with different size of child arrays is called jogged array

**// int**n[][]=**newint**[][8]; we must specify parent locations

Collection:

Customcollection:

1. First we take object array with limited size
2. If we add objects in that growable array we can write a method add() in that first before adding element to array we can check the size of array and capacity of array equal we can increment capacity()
3. In increment capacity we can take temporary array object with twice of capacity
4. Next move elements from old array to new array .
5. Then pointing to new array to old array.
6. Then store new elements.

**package** com.nare;

**public** **class** CustomCollection {

**private** Object[] ob = **new** Object[10];

**private** **int** index = 0;

**public** **void** add(Object o) {

**if**(size()==capacity()) {

incrementCapacity();

}

ob[index]=o;

index++;

}

**private** **void** incrementCapacity() {

Object[] temp = **new** Object[capacity()\*2];

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

temp[i]=ob[i];

}

ob=temp;

}

**public** **int** capacity() {

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

**return** ob.length;

}

**public** **int** size() {

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

**return** index;

}

**public** Object get(**int** i) {

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

**return** ob[i];

}

**public** **void** replace(**int** i,Object o) {

ob[i]=o;

}

@Override

**public** String toString() {

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

StringBuilder b=**new** StringBuilder();

b.append("[");

**for**(**int** i=0;i<size();i++) {

b.append(ob[i]);

b.append(",");

}

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

**int** start=b.lastIndexOf(",");

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

**if**(start!=-1) {

b.deleteCharAt(start);

}

b.append("]");

**return** b.toString();

}

}

**package** com.nare;

**public** **class** CustomCollectionTest {

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

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

CustomCollection c=**new** CustomCollection();

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

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

c.add("1");

c.add("2");

c.add("3");

c.add("4");

c.add("5");

c.add("6");

c.add("7");

c.add("8");

c.add("9");

c.add("10");

c.add("11");

c.add("12");

c.add("13");

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

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

System.***out***.println(c.get(2));

c.replace(0,"balu");

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

}

}

Op: 10

0

20

13

3

kdkddk

33

[balu,2,3,4,5,6,7,8,9,10,11,12,13]