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]

23-12-2021

Print : print method is print the output in sameline

Println : this method is cursor go with next line and print the output.

Comments:

// Single line comments

/\* \*/multiple lines comments

/\*\* \*\*/ documents comments

Accesibility modifiers:

1. Private : we can access with in the class only.
2. Protected : permissions only with in that package only.
3. Public : we can access in anywhere in the application.

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

Datatype: Store data in temporary in computer through program.

Variable : variable is named memory location used to store data

Transient variable: these are class level variables . if we declare variable is transient varible these variables are not store in file in the concept of serialization. If we declare transient variable in local is illegal local is not part of object.

Volatile variable: if we daclare varible is volatile to tell to jvm that variable is not modified concurrently by multiple threads.

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

1. Bootstrap class loader: it is responsible to call the java library class file i.e rt.jar classes
2. Extension class loader: its call user defined classes and java library class files from extension path ie ext folder
3. Application class loader: its called user defined classes from application classpath.

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

Static members: static members using in below situations

1. Import statements
2. Class level variable
3. Method level
4. Class level block
5. Class level inner class

Static members get memory at time of class loading by default

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

Main method : mediator between jvm and developer

Public : main method is called jvm from out side package.

Static: why because it should be identified at the time of class loading . we getting memory without object creation.

Void : jvm does not want return value so main method is void return type.

String[] : it allows multiple strings in array format.

Oop principles :

Encapulation: the process of creating a class hiding internal data from outside .and accessing through publicly exposed methods.

We are taking the variables as private it access only through setter and getters. So we can prevent un otherised access and security

Autoboxing : converting primitive type to wrapper class object automatically is called

Auto unboxing: converting wco to primitive type is called

Exception : Exception is a class that is used to handle runtime errors.

Checked Exception: if the exception is thrown by throw keyword that Exception is checked by compiler then that exception is called checked exception. Ex: Error and Runtime Exception are the checked exception so programmer need not to catching or reporting is optional.

Un Checked Exception: if the exception is thrown by throw keyword that Exception is not checked by compiler then that exception is called un checked exception. Ex: Throwable ,Exception and its direct classes are unchecked Exception so programmer must need to catching and reporting is mandator

Throw : is used to throw exception manually from a method or constructor. Mostly we are throwing checked exceptions only.

Thows : is used to report that raised exception to this method caller. It is mandatory to checked exceptions to report.