. Abstract class cannot be instantiated.

ex8.method2(); // // outside package protected is not possible to access via object

method2(); //acessed directly

//objects and its instance variable resides in heap

//methods and local variable resides in stack

//shorthand operators performs implicit cast

byte hij = 6;

byte jkl = 8;

jkl += hij;

//change in one object result in change in other, but strings also objects, but they have special treatrment. Change in one will nt affect other.

since different objects are created in memory

//withous shadowing(using this. ) will not the change instance varibale value,

it remains same in both static and non static instance variable

/\*Child2[] ch2 = {new Parent()}; //superclass to subclass is not possible

// wrapper methods

. primitive types to objects

int ab = 5;

Integer ib = new Integer(ab);

.Objects to primitives

Integer ib = new Integer(5) or Integer ib = new Integer("5")

int ab = ib.intValue();

.String to primitives

double d1 = Double.parseDouble("789.456");

.String to wrapper

Double d1 = Double.valueOf("3.14");

//priority = primitive -> wrapper->var args

byte a = 6;

add(a) // will call to add(int a) even though add(Byte a) is there.

The priority is to nearest primitives, if not coressponding warpper ,if not var args

//garbage collections

Runtime rt = Runtime.getRuntime();

rt.totalMemory();

rt.freeMemory();

rt.gc(); or System.gc();

//overloaded method calling is always based on reference object(LHS)

//overridden method calling is always based on instance object(RHS), even then LHS reference class must contain the method

//Static method cannot be overridden

Serializable

To save the object

.implements serilazable

.write to file

.FileOutputStream

.ObjectOutputStream

. writeObject(Object)

.read from file

.FileInputStream

.ObjectInputStream

.(typecast)readObject()

.transient

.not to save

ex : class Cat implements serializable {

transient Animal ab; // Animal object data will nt be saved as part of animal

.

}

whjen restored, Animal object data restored with default values

Not the values which was exist during Cat serilization

. To save at particular situation, the use below two methods

.private void writeObject(ObjectOutputStream os)

.private void readObject(ObjectInputStream is)

.If superclass implements serilazable, by default all its subclasses implements serializable

.staic variables serialzation are nt possible

//serializations

public class Example3 {

public static void main(String[] args) throws IOException, ClassNotFoundException {

Student s1 =new Student(211,"ravi");

FileOutputStream fout=new FileOutputStream("f.txt");

ObjectOutputStream out=new ObjectOutputStream(fout);

out.writeObject(s1);

out.flush();

System.out.println("success");

ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));

Student s=(Student)in.readObject();

System.out.println(s.id+" "+s.name+" "+s.company);

in.close();

}

}

class Student implements Serializable{

int id;

String name;

// Student2 s2 = new Student2(211, "ravi"); //gives NotSerializableException during runtime

//to avaoid we should serialize Student2 also

static String company;//it won't be serialized

transient private int salary;

public Student(int id, String name) {

this.id = id;

this.name = name;

this.company="SSS ";

}

}

class Student2 {

int id;

String name;

public Student2(int id, String name) {

this.id = id;

this.name = name;

}

}

Externalizable (which extends Serializable and forces two methods to implement)

@Override

public void writeExternal(ObjectOutput out) throws IOException {

out.writeInt(id);

out.writeObject(name+"xyz");

out.writeObject("abc"+gender);

}

@Override

public void readExternal(ObjectInput in) throws IOException,

ClassNotFoundException {

id=in.readInt();

//read in the same order as written

name=(String) in.readObject();

if(!name.endsWith("xyz")) throw new IOException("corrupted data");

name=name.substring(0, name.length()-3);

gender=(String) in.readObject();

if(!gender.startsWith("abc")) throw new IOException("corrupted data");

gender=gender.substring(3);

}

//externizable

public class Example4 {

public static void main(String[] args) {

String fileName = "person.ser";

Person person = new Person();

person.setId(1);

person.setName("Pankaj");

person.setGender("Male");

try {

FileOutputStream fos = new FileOutputStream(fileName);

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(person);

oos.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

FileInputStream fis;

try {

fis = new FileInputStream(fileName);

ObjectInputStream ois = new ObjectInputStream(fis);

Person p = (Person)ois.readObject();

ois.close();

System.out.println("Person Object Read="+p);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

class Person implements Externalizable{

private int id;

private String name;

private String gender;

@Override

public void writeExternal(ObjectOutput out) throws IOException {

out.writeInt(id);

out.writeObject(name+"xyz");

out.writeObject("abc"+gender);

}

@Override

public void readExternal(ObjectInput in) throws IOException,

ClassNotFoundException {

id=in.readInt();

//read in the same order as written

name=(String) in.readObject();

if(!name.endsWith("xyz")) throw new IOException("corrupted data");

name=name.substring(0, name.length()-3);

gender=(String) in.readObject();

if(!gender.startsWith("abc")) throw new IOException("corrupted data");

gender=gender.substring(3);

}

RegExp

.Pattern.compile("pattern")

.p.matcher("input")

.find()

while matcher.find()

.m.start()

starting index

.m.group()

group of matched characters

The lookingAt() method only matches the regular expression against the beginning of the text,

whereas matches() matches the regular expression against the whole text

start() and end() will give the indexes into the text where the found match starts and ends.

Actually end() returns the index of the character just after the end of the matching section

By calling reset() the matching will start from the beginning of the text again

Scanner

Scanner(System.in()

String token = s.findInLine("pattern")

Strings

String constant pool

.If same object exist, it will nt create new object

rather it points to same object

String s1 = "abc";

String s2 = "abc";

both s1 and s2 points to abc, one object

s1 == s2 and s1.equals(s2) both returns true

Non pool(heap)

String s1 = new String("abc");

String s2 = new String("abc");

s1 and s2 points to two different location

s1 == s2 returns false

s1.equals(s2) returns true provided equals method is overridden

.String buffer v/s String builder

String buffer -> thread safe, methods are synchronized, slower

String builder -> not thread safe, methods are nt synchronized, faster

File

.File

createNewFile()

isFile()

isDirectory()

mkdir()

delete()

renameTo()

list()

exists()

FileReader

read()

BufferReader()

read()

readLine()

FileWriter()

write()

flush()

BufferWriter()

write()

newline()

flush()

PrintWriter()

print()

println()

printf()

format()

flush()

system.console()

readLine()

readPassword()

Dates

java.util

Date

Calendar

Locale

java.text

DateFormat

NumberFormat

Abstarct

Calendar

DateFormat

NumberFormat

Non abstract

Date

Locale

1) Date

less prefer

takes miliseconds, operation performed only on miliseconds

2) Calendar

Calendar.getInsatnce()

Subclass hierarchy methods are classed due to abstract

more prefer

3) DateFormat

Define the DateFormat

pass the Date object to it

4)Locale

To get dates in international format

Define DateFormat

pass the locale object

5)NumberFormat

To format the numbers, dates o/p

super() -> to call parent class constructor

this() -> to call same class constructor

super.method -> to call parent class method

this.method -> to call same class method

.system.out.printf

%[arg\_index][flags][width][.precision]conversion character

. without tostring overridden, syso of object o/p is packagename.classname@hashcode

. == returns true if two object refernce ponts are same, equals returns true only if two objects refernces and contents are actually equal

. syntax of toString : public String toString() { returns "...."; }

public class C\_EqualsOveride2 {

protected long employeeId;

protected String firstName;

protected String lastName;

public boolean equals(Object o){

if(o == null) return false;

if(!(o instanceof C\_EqualsOveride2)) return false;

C\_EqualsOveride2 other = (C\_EqualsOveride2) o;

if(this.employeeId != other.employeeId) return false;

if(! this.firstName.equals(other.firstName)) return false;

if(! this.lastName.equals(other.lastName)) return false;

return true;

}

}

. syntax of hashcode :public int hashCode() {

return value\*24;

}

. hashcode() gives exact location of object, to check equality equals method should be used

.Because two objects can have same hashcode, to search for it we should use equals method to find the actual object from the location returned by hashcode

. Dont use transient incase of equals and hashCode overidden

. File IO

.system.in, sytem.out, system.err

. Byte stream

InputStream(read), OutputStream(write), BufferdInputStream, BufferedOutputStream, DataInputStream, DataOutputStream

.Character stream

Reader,Writer

. Collection is interface, Collections is class with util methods like sort

. Collection interface

. set interface : No duplicates

. HashSet -> No sort, no ordering

. LinkedHashSet : order by insertion

. TreeSet : Ordered and sorted by natural order

public class M\_TreeSetCreationWithComparator {

public static void main(String a[]){

TreeSet<String> ts = new TreeSet<String>(new MyComp());

ts.add("RED");

ts.add("ORANGE");

ts.add("BLUE");

ts.add("GREEN");

System.out.println(ts);

}

}

class MyComp implements Comparator<String>{

@Override

public int compare(String str1, String str2) {

return str1.compareTo(str2);

}

}

public class N\_TreeSetSubsetExamples {

public static void main(String a[]){

TreeSet<String> ts = new TreeSet<String>(new MyStrComp());

ts.add("RED");

ts.add("ORANGE");

ts.add("BLUE");

ts.add("GREEN");

ts.add("WHITE");

ts.add("BROWN");

ts.add("YELLOW");

ts.add("BLACK");

System.out.println(ts);

Set<String> subSet = ts.subSet("GREEN", "WHITE");

System.out.println("sub set: "+subSet);

subSet = ts.subSet("GREEN", true, "WHITE", true);

System.out.println("sub set: "+subSet);

subSet = ts.subSet("GREEN", false, "WHITE", true);

System.out.println("sub set: "+subSet);

}

}

class MyStrComp implements Comparator<String>{

@Override

public int compare(String str1, String str2) {

return str1.compareTo(str2);

}

}

. List interface: Duplicates allowed

. ArrayList : insertion and deletion slower, Non Synchronized

. Vector : Synchronized

. LinkedList : search slower

. Map interface :

. HashMap : one key null, many null values

. Hashable : no key null

. LinkedHashMap : sorted by insertion order

. TreeMap : sorted and ordered by natural order

.Comparable interface

compareTo method name, one argument

return title.compareTo(o.title)

.once defined, sort can be performed one way

.Comparator interface

compare method with two arguments

return arg1.title.compareTo(arg2.title)

. sort can be performed with based on data

.Arrays.asList() : for connverting arrays to list

.toArray : for converting list to arrays

.changes in one , changes to other also

.size has to be same, once we assigned to data structure, we cant insert .Only we can replace. Because arrays are fixed size even though list is not

. iterators for Collection

. Set add returns true if not exist, returns false if exist

.

.TreeSet Methods

. lower, ceiling, higher, floor, pollFirst, pollLast, descendingSet, headSet, tailSet, subSet

System.out.println(st1.lower(23)); //less than 23

System.out.println(st1.ceiling(22)); //22 or less than that

System.out.println(st1.higher(23)); //greater than 23

System.out.println(st1.floor(22)); //22 or greater than 23

System.out.println(st1.descendingSet()); //reverse order

System.out.println(st1.headSet(23)); //top most elements of 23

System.out.println(st1.tailSet(23)); // 23 and down elements of 23

System.out.println(st1.subSet(17, 45)); // 17 and between 17 and 45 excluding 45

System.out.println(st1.pollFirst());

System.out.println(st1.pollLast());

.TreeMap Methods

.lowerkey, ceilingKey, higherKey, floorKey, pollFirstEntry, pollLastEntry, descendingMap, headMap, tailMap, subMap

.Collection : poll(removes and returns element)

: peek(returns element)

.Priority Queue methods :

Offer, peek, poll

.Sort function : spaces first, caps next and lower case last

.Generic types :

=======================

.Arrays => any subtype refernce can be passed to supertype, but casting required while retrieving

Animal[] arr1 = {new animal(), new Dog()} // legal where dog extends Animal

.Collections => Reference type has to be same

ex : List<Dog> st1 = new ArrayList<Dog>();

List<Animal> st1 = new arrayList<Dog>(); //illegal even though Dog extends Animal

//Storing wrong type of abjects in arrays gives runtime ArraysStoreExceptions

//Storing wrong type of objects in collections gives compiler error

.Collections add function, we can add subtype objects

ex : animal.add(new Dog());

.public void refer(List<? extends Animal> animals)

animal type or its subtypes

.public void refer(List<super extends Dog> dogs)

dog type or its parents(top in hierarchy)

.List<?> is equalent to List<? extends Object> are identical

.List<Object> only Object can be passed , not other even though evrything extends Object

refer(dogs) not allowd, refer(Obj) allowed where Object obj;

.Class level indication or method level indication needed for generics

ex : public class test<T,S> { //class level

}

ex : public <T,S> void refer(T type1, S type2) { //method level, before return type

}

.see last two examples

//Arrays to treeset

Arrays to list (using Arrays.asList(array), new Treeset<String>(arraylist)

//List to array

list.toArray();

//changes in one collection data structure result in other data structure during conversions

//finding duplicates in array

. for each array elements, add to tree set

if tree set add returns false, then it is duplicate

.javac Xlint:unchecked

.used to get more details about warnings to avoid runtime exceptions

. If object1 and object2 are equal according to their equals() method, they must also have the same hash code.

If object1 and object2 have the same hash code, they do NOT have to be equal too.

. Both TreeMap and TreeSet are non synchronized Collection, hence can not be shared between multiple threads.

You can make both TreeSet and TreeMap synchronized by wrapping them into Synchronized collection by calling

Collections.synchroinzedMap()

Inner Class

. compiler creates two class files, one for outerclass and one for inner class

. no concept of static inner class, inner class acts as static memebr of outer class

.Topics are

.inner class

.this usgae

.annonymous classes(extends(inheritance) and implement(interface))

.static inner classes

. Outerclass variable and methods are visible to inner class

. OuterClass oc = new OuterClass();

InnerClass ic = oc.new InnerClass();

or

InnerClass ic = new OuterClass().new InnerClass();

.Static inner class

InnerClass ic = new OuterClass().InnerClass();

. Inside inner class

this - > inner class object

OuterClass.this -> outerclass object

. Aninymous inner class for class or for interface

D\_AnnonymousExtendClass ac = new D\_AnnonymousExtendClass() {

public void display() {

System.out.println("Oueriding anonymous syper class display method");

}

public void other() {

System.out.println("test");

}

};

class AnnonymousInterface {

AnnonymousClassInterface ac = new AnnonymousClassInterface() {

public void display() {

System.out.println("Implementing interface display method");

}

public void other() {

System.out.println("test");

}

};

.......

}

Threads

.Thread synchronize examples

.Thread deadlock examples

.Thread interaction examples ( wait, notify, notifyall())

Treads creation 2 ways

.By extending java.lang.Thread

Class MyThread extends Thread {

public void run() {

}

}

MyThread th = new MyThread();

th.start();

.This is less prefer, because we cant extend any other class

.By implememnting java.lang.Runnable

Class MyRnnable implements Runnable {

public void run() {

}

}

MyRunnable r = new MyRunnable();

Thread t = new Thread(r);

t.start();

.This is more prefer, we can extend other class

. Once the thread is started, start() on thread again causes IllegalthreadStateException

. Thread states

New

Runnable

Running

waiting/blocked.sleeping

Dead

.Thread methods

.sleep() takes arguments in miliseconds

.minimum duration that thread sleeps. it might sleep more than that.

.yeild()

. Moves the thread to Runnable state and makes other thread running which has same priority.

.join()

.ex : b.join()

If A is ruuning, calling b.join makes b to run and A to be in runnable state. Once b completes, a resumes his execution

.setPriority() -> 1 to 10, default is 5

.setName() -> to set name for thread

.Thread.currentThread().getName() -> to get thread name

.Synchronize

. require when two thread invokes run which changes common variable

. ex : Bank withdrawal based on balance, when two thread checks .

balance exist. If one thread withdraws all amount,

other tread checkbalance sucess and tries to withdraw.

It gets fail

.synchronize

.Method synchronize

.private synchronized ..() {

}

.Bloock of code synchronize

.Non static method synchronize

public void method() {

....

synchronized(this);

............

}

.static method synchronize

public static void method() {

....

synchronized(Classname.class);

............

}

.each thread gets seperate copy of local variable copies, no worry about it.

.static and non static fields, use seperate synchronized getter and setter methods to set and get the value

.Thread interaction

.wait : makes thread to wait until it gets notify message

.notify : notify the waiting thread

,notifyall : notifies all waiting threads.

* **Generics and Collections**
* toString(),equals(),hashcode()
  + returns hashcode of the object
  + each class extends object by default, so overide the toString in class how object has to be printed
  + When you really need to know if two references are identical, use ==. But when you need to know if the objects themselves (not the references) are equal, use the equals() method.
  + In hashtable, if we are using objects as key : so unless you override equals(), two objects are considered equal only if the two

references refer to the same object

* + Remember ***that the*** equals()***,*** hashCode()***, and*** toString() ***methods are***

***all*** public***. The following would not be a valid override of the*** equals() ***method, although it might appear to be if you don’t look closely enough during the exam:***

class Foo { boolean equals(Object o) { } }

* + And ***watch out for the argument types as well. The following method is an***

***overload, but not an override of the*** equals() ***method:***

class Boo { public boolean equals(Boo b) { } }

public class EqualsTest {

public static void main (String [] args) {

Moof one = new Moof(8);

Moof two = new Moof(8);

if (one.equals(two)) {

System.out.println("one and two are equal");

}

}

}

class Moof {

private int moofValue;

Moof(int val) {

moofValue = val;

}

public int getMoofValue() {

return moofValue;

}

public boolean equals(Object o) {

if ((o instanceof Moof) && (((Moof)o).getMoofValue()

== this.moofValue)) {

return true;

} else {

return false;

}

}

}

* + The hashcode tells you only which bucket to go into,

but not how to locate the name once we're in that bucket

* + ***In*** real***-life hashing, it’s not uncommon to have more than one entry in a***

***bucket. Hashing retrieval is a two-step process.***

***1. Find the right bucket (using*** hashCode()***)***

***2. Search the bucket for the right element (using*** equals() ***).***

* + So one more ***time***: If two objects are equal, their hashcodes must be equal as well.
  + ***So in order for an object to be located, the search object and the object in***

***the collection must have both identical hashcode values and return true for the***

***equals() method.***

***Conditions Required Not required***

**Condition Required Not Required (But Allowed)**

x.equals(y) == true x.hashCode() == y.hashCode()

x.hashCode() == y.hashCode() x.equals(y) ==

true

x.equals(y) == false No hashCode()

requirements

x.hashCode() != y.hashCode() x.equals(y) == false

* + Bottom line: transient variables can really mess with your equals() and

hashCode() implementations. Keep variables non-transient or, if they must

be marked transient, don't use them to determine hashcodes or equality.

* **Development**
* Compilation
  + javac -help

javac -classpath com:. -g Foo.java Bar.java

The first invocation doesn't compile any files, but prints a summary of valid

options. The second invocation passes the compiler two options (-classpath,

which itself has an argument of com:. and -g), and passes the compiler two .java

files to compile (Foo.java and Bar.java). Whenever you specify multiple options

and/or files they should be separated by spaces.

* + javac -d classes source/MyClass.java
    - compiles and puts the class in destination folder classes
  + javac -d ../classes com/wickedlysmart/MyClass.java

myProject

|

|--source

| |

| |--com

| |

| |--wickedlysmart

| |

| |--MyClass.java

|

|--classes

| |

| |--com

| |

| |--wickedlysmart

| |

| |-- (MyClass.class goes here)

Note : classes directory has to be exist, if not

java:5: error while writing MyClass: classes/MyClass.class

(No such file or directory)

* Execution
  + You must specify exactly one class file to execute, and

the java command assumes you're talking about a .class file, so you don't specify

the .class extension on the command line

* + java -DmyProp=myValue MyClass x 1

this command can be read as "Create a *system*

*property* called myProp and set its value to myValue. Then launch the file named

MyClass.class and send it two String *arguments* whose values are x and 1."

* + Accessing system properties

import java.util.\*;

public class TestProps {

public static void main(String[] args) {

Properties p = System.getProperties();

p.setProperty("myProp", "myValue");

p.list(System.out); //lists all

}

}

java -DcmdProp="cmdVal take 2" TestProps

o/p contains both set from command line and set by code and other system properties also

* + The following are all legal declarations for main():

static public void main(String[] args)

public static void main(String... x)

static public void main(String bang\_a\_gong[])

* + java and javac

In the case that their search lists contain two or more files with the

same name, the first file found will be the file that is used.

A classpath can be declared as a command-line option for either java or

javac. *Classpaths declared as command-line options override the classpath declared*

*as an environment variable, but they persist only for the length of the invocation.*

*Most of the path-related questions on the exam will use Unix conventions.If you are a Windows user, your directories will be declared using backslashes (\) and the separator character you use will be a semicolon (;).*

*The way to tell java or javac to search in the current directory is to add a dot (.) to the classpath:* java or javac to *also* search for class files in the current

directory

*-classpath /com/foo/acct:/com/foo:****.***

It's also important to remember that classpaths are searched from left to right.

Therefore in a situation where classes with duplicate names are located in several

different directories in the following classpaths, different results will occur:

-classpath /com:/foo:.

is not the same as

-classpath .:/foo:/com

In order to find a class in a package, you have to have a directory in your classpath

that has the package's leftmost entry

* + JAR stands for Java Archive.
    - jar -cf MyJar.jar myApp
      * create a JAR file called MyJar.jar and it will contain the

myApp directory and myApp's entire subdirectory tree and files

* + - jar -tf MyJar.jar
      * contents of the JAR file
    - ***The jar command*** creates ***the*** META-INF ***directory automatically.***
    - ***The jar command creates the*** MANIFEST.MF ***fi le automatically.***
    - ***The jar command won’t place any of your fi les in*** META-INF/***.***
    - To compile class file in jar file
      * javac -classpath ws/myApp.jar UseStuff.java
    - import ***statement can import only a single package.***
      * ***we can’t say*** import java.\*;
  + Static imports

Before static imports:

public class TestStatic {

public static void main(String[] args) {

System.out.println(Integer.MAX\_VALUE);

System.out.println(Integer.toHexString(42));

}

}

After static imports:

import static java.lang.System.out; // 1

import static java.lang.Integer.\*; // 2

public class TestStaticImport {

public static void main(String[] args) {

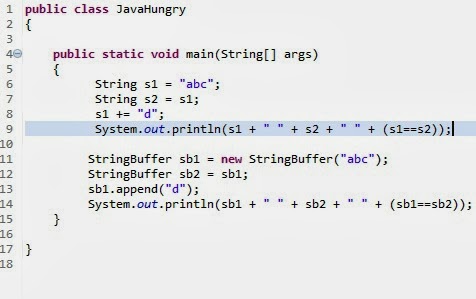
out.println(MAX\_VALUE); // 3

out.println(toHexString(42)); // 4

}

}

* You must say import static; you can't say static import.
* if you do a static import for both the Integer class and the Long class, referring to MAX\_VALUE will cause a compiler error, since both Integer and Long have a MAX\_VALUE constant, and Java won't know which MAX\_VALUE you're referring to.



***Answer >>***  
 ***Compile :  Yes***  
***Output  :    abcd abc false***  
***abcd abcd  true***