**Assert** (18)

**Boxing / Un-Boxing / Auto-Boxing** (25)

**Check Java Version in Unix** (1) $ java –version

**Calendar class** (13)

**Collections**(66-74)

**Class class** (32)\*

**Generics** (31-39, 47-51,71)\*

--Generic Class - 32

--Generic Class (Multiple Types) - 34

--Problems and Restrictions (35)

--Method creation (47)

--Constructor (49)

--Interface(50)

**Enumeration** (53-58)

**File Class**  63 would give you more info about the File class.  (214)

**Factory Methods** (13)

**Javac** (21)

**Log Class** (53)

**Mainless Program** (5)

**My Methods**

--print (4)

--die (4)

--loadFile(10)

--continueYN (12)

-randomNumberGenerator(20)

**New in Java 7**

--write into file (1)

--use underscore to separate part of your integers (2)

--switch can now use Strings (3)

--combine all exceptions together (3)

-ownership of a file (41)

**PrintWriter** (6, 11, 14)\* PrintWriter class, then go to page 13

**Primitive vs Reference** (24)\*

-pass by value vs pass by reference

**Serialization** (60-62) 39, would help you about object serialization

**Subtyping** (31)

**System.getProperty**(58)\*

**swap(a,b)** (63,65)

**ToolKit Class** (17)

**Zipping Files** zipping a file, go to the page 33

**Questions:**

Add data into file (11)

assert question (26)

Closure(63)

Check Java version (1) or (58)

Create unique numbers using Binary (42)

Comparable vs Comparator (50)  
New in Java 7 (1)

Ownership of a file (41)

How to read a file from the web (url)? (17)

What is inside Iterable<E> and Iterator<E> (27)

Version of Java (58)\*

PRINT WRITER Example:\*

/\*\*  
 \* @Program uses write method to write single character in to   
 \* the file specified.  
 \* Write3.java   
 \* Author:-RoseIndia Team  
 \* Date:-7-july-2008  
 \*/  
  
**import**java.io.File;  
**import**java.io.IOException;  
**import**java.io.PrintWriter;  
  
**public class**Write3 {  
  **public static void**main(String args[]) **throws**IOException {  
    File witches = **new**File("c:\celina.doc");  
    PrintWriter out = **new**PrintWriter(witches);  
  
    out.write('1');  
    out.write('2');  
    out.write('3');  
  
    out.println("");  
  
    out.write('1');  
    out.write('2');  
    out.write('3');  
  
    out.checkError();  
  }

Class method\*

|  |  |
| --- | --- |
| [String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) | [**getName**](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Class.html#getName())()            Returns the name of the entity (class, interface, array class, primitive type, or void) represented by this Class object, as a String. |

GENERICS\*

Let's test your understanding of generics. Is the following code snippet legal?

List<String> ls = new ArrayList<String>(); // 1

List<Object> lo = ls; // 2

Line 1 is certainly legal. The trickier part of the question is line 2. This boils down to the question: is a List of String a List of Object. Most people instinctively answer, "Sure!"

Well, take a look at the next few lines:

lo.add(new Object()); // 3

String s = ls.get(0); // 4: Attempts to assign an Object to a String!

Here we've aliased ls and lo. Accessing ls, a list of String, through the alias lo, we can insert arbitrary objects into it. As a result ls does not hold just Strings anymore, and when we try and get something out of it, we get a rude surprise.

The Java compiler will prevent this from happening of course. Line 2 will cause a compile time error.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Three ways to invoke a method:**

1. Instance Method

File f = new File(“etc/xyz”);

if(f.exists())

{

}

2. Static or class method

x = Math.sqrt(49);

3. Inheritance

class Doit extends Abc

{

}

Recall

– inheritance allows you to design a new class from an existing class

- child class inherits characteristics of the parent (method, data)

-variables and methods that are private in the parent class cannot be referenced by name in the child class

-child class does not inherit the constructor; needs to create its own

-child can use parent’s constructor to set up

--super reference to call the parent’s constructor or method or variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Read write to file** (10)

To read input we are using [File.](http://download.oracle.com/javase/6/docs/api/index.html?java/io/File.html)

File f = new File("myFile");  
Scanner sc = new Scanner(f);

For output we are using [PrintWriter](http://download.oracle.com/javase/6/docs/api/index.html?java/io/PrintWriter.html)

PrintWriter pw = new PrintWriter("hisfile");

PrintWriter uses the methods print(), println(),printf().

pw.println("Hello Java");

Hello Java will go in to the file.

Appending a file is tricky. The easiest way to append is by using [FileWriter](http://download.oracle.com/javase/6/docs/api/index.html?java/io/FileWriter.html) and PrintWriter.

FileWriter fw = new FileWriter("herfile", True);  
 PrintWriter pw = new PrintWriter(fw);

Where boolean is either true to append or false for don't append. Make sure you become familiar with printf()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**My Methods**

We need to display error message and exit. We will learn to create our own exceptions but for now we will use the method die(). It should be one of the first method in your programs.

public static void die(String msg)  
 {  
 System.err.println(msg);  
 System.exit(1);  
 }

Example:

if(num < 0) die("negative number");

But what if you need to create multiple versions of die()? If you have 0 or more variables you could overload but there is a better way.

public static void die(String ... msg)  
 {  
 // msg is now treated as an array  
 if(msg.length == 0)  
 {  
 System.err.println("Error Happened");  
 }  
 else  
 {  
 for(String s : msg)  
 {  
 System.err.println(s);  
 }  
 }  
 System.exit(1);  
 }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Factory Method**

abstract class DoIt  
{  
    public static DoIt getInstance()  
    {  
       return(new WillDo());  
    }

public abstract void sayHello(); // no implementation!!

}

class WillDo extends DoIt  
{  
    public  void sayHello()  
    {  
       System.out.println("Hi, juju");  
    }  
  
}

public class DemoDoit  
{  
  
  public static void main(String args[])  
  {  
    DoIt di=DoIt.getInstance();  // create an object of abstract class by factory method  
    di.sayHello();  
  }  
  
}

\*\*\*\*\* Interface doesn't have any concrete method. no implementation  
why use interface:  
  Java, you cannot inherit(extend) more than one class, but you can add as many interface as needed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Boxing / Unboxing / Swap**

1. What are boxing and unboxing?  
  
  - Ex. What is "Integer" for? when there is 'int'.  
    : convert the number into an object.  
  - Thus, all the primitive data types have equivalent classes.  
  
   Integer x=new Integer(100); // boxing  
   int y=x.intValue();          // unboxing  
  
  - In newer versions, boxing/unboxing done automatically.  
    ex) x++;  //works  
        x+= x/2   ;  // works but it will be slow.  
  
2. How to do the jobs need the pointer/reference?  
     
  1) using a class  
   
     class Num  
    {  
       int x;  
       int y;  
    }  
  
    Num n=new Nums();  
    n.x=9;  
    n.y=5;  
    swap(n);  
  
    public void swap(Nums a){  
       int temp;  
       temp=a.x;  
       a.x=a.y;  
       a.y=temp;  
    }

**Generics**

There may be times when you'll want to restrict the kinds of types that are allowed to be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.

class Stats <T extends Number>   
// this 'extends' is not about inheritance  
// Numeric types: Integer, Double, Short,Byte,Long,Float   
// Number is a super class of all above and an abstract class  
// Number class has 'valueOf()' method  
// T will not accept String, Boolean..etc.  
{  
    T nums[];  
    Stats(T o[]){nums=o;}  //constructor  
    double average(){  
    double sum = 0.0;  
    for(T n:nums) sum += n; // to be continued  
    return(sum/nums.length);  
  
    }  
}

//class Gen<T extends MyClass & MyInterface >{   
          // data type should be MyClass or its subclasses  
          // also should implements MyInterface  
          // & MyInterface & MyInterface2 & MyInterface3>  
          // or <T extends MyInterface> is possible.  
          // or <T MyInterface> is possible.  
          // interface is also extendable.

Wildcards

Note: It's also possible to specify a lower bound by using the super keyword instead of extends. The code <? super Animal>, therefore, would be read as "an unknown type that is a supertype of Animal, possibly Animal itself". You can also specify an unknown type with an unbounded wildcard, which simply looks like <?>. An unbounded wildcard is essentially the same as saying <? extends Object>.

public static void main(String args[])  
    {  
         String str = System.getProperty("java.version");  
         int version = Integer.parseInt(str.substring(2,3));  
         if(version > 6) fixIt(60);  
         else  fixNum(60);       
    }  
  
    public static void fixIt(int x)  
    {  
         System.out.println("fixIt : " + x);  
    }  
    public static void fixNum(int x)  
    {  
         System.out.println("fixNum : " + x);  
    }

}

**Version of Java**

|  |  |
| --- | --- |
| java.version | Java Runtime Environment version |
| java.vendor | Java Runtime Environment vendor |
| java.vendor.url | Java vendor URL |
| java.home | Java installation directory |
| java.vm.specification.version | Java Virtual Machine specification version |
| java.vm.specification.vendor | Java Virtual Machine specification vendor |
| java.vm.specification.name | Java Virtual Machine specification name |
| java.vm.version | Java Virtual Machine implementation version |
| java.vm.vendor | Java Virtual Machine implementation vendor |
| java.vm.name | Java Virtual Machine implementation name |
| java.specification.version | Java Runtime Environment specification version |
| java.specification.vendor | Java Runtime Environment specification vendor |
| java.specification.name | Java Runtime Environment specification name |
| java.class.version | Java class format version number |
| java.class.path | Java class path |
| java.library.path | List of paths to search when loading libraries |
| java.io.tmpdir | Default temp file path |
| java.compiler | Name of JIT compiler to use |
| java.ext.dirs | Path of extension directory or directories |
| os.name | Operating system name |
| os.arch | Operating system architecture |
| os.version | Operating system version |
| file.separator | File separator ("/" on UNIX) |
| path.separator | Path separator (":" on UNIX) |
| line.separator | Line separator ("\n" on UNIX) |
| user.name | User's account name |
| user.home | User's home directory |
| user.dir | User's current working directory |

String ver = System.*getProperty*("java.version", "6.1");

System.*out*.println(ver);

//ver = ver.substring(2, 3);

//System.out.println(ver);

Character c = ver.charAt(2);

System.*out*.println(c);

**class** Zip

{

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*unzip()\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **void** unzip(String zipfile)

{

**try**

{

String inFilename = zipfile;

ZipInputStream zis = **new** ZipInputStream(**new** FileInputStream(

inFilename));

ZipEntry ze = zis.getNextEntry();

String outFilename =

zipfile.substring(0, zipfile.lastIndexOf(".zip"));

FileOutputStream os = **new** FileOutputStream(outFilename);

**byte** buf[] = **new** **byte**[1024];

**int** len;

**while**((len = zis.read(buf)) > 0) os.write(buf, 0, len);

os.close();

zis.close();

}**catch**(IOException e) {}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*zip()\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **void** zip(String fname)

{

**byte** buf[] = **new** **byte**[1024];

**try**

{

String outFilename = fname + ".zip";

ZipOutputStream zos = **new** ZipOutputStream(**new** FileOutputStream(

outFilename));

FileInputStream fis = **new** FileInputStream(fname);

zos.putNextEntry(**new** ZipEntry(fname));

**int** len;

**while**((len = fis.read(buf)) > 0) zos.write(buf, 0, len);

zos.close();

fis.close();

}**catch**(IOException e){}

}

}

**Pass by Value (262)**

Means that primitive data types are passed by value, which means that only a copy of argument’s value is passed into a parameter variable. A method’s parameter variables are separate and distinct from the arguments that are listed inside the parentheses of a method call. If a parameter variable is changed inside a method, it has no effect on the original argument.

**printf Method** (164)

System.out.printf(“String %s %d %.4f\n”, name1, value1, value2);

**Parse (96) (string- to- numbers)**

String num;

Scanner keyboard = new Scanner(System.in);

num = keyboard.nextLine();

int number;

number = Integer.parseInt(num);

class

{  
    public static void main(String args[])  
    {  
         String str = System.getProperty("java.version");  
         int version = Integer.parseInt(str.substring(2,3));  
         if(version > #) method1(##);  
         else  method2(##);     //n  
    }  
  
    public static void fixIt(int x)  
    {  
         System.out.println("fixIt : " + x);  
    }  
    public static void fixNum(int x)  
    {  
         System.out.println("fixNum : " + x);  
    }  
      
}

2) Collection i/f  
  
interface Collection<T>  
{  
boolean add(E obj);  
boolean addAll(Collection <? extends E> c);   // E or E's subclass  
void clear(); // removes all elements  
boolean contains(object obj); // this methods doesn't handle generic. use casting instead.  
boolean containsAll(Collection <?> c);  
boolean equals(object obj);  
booleab isEmpty();  
Iterator<E> iterator(); // to create the object of iterator  
booleab remove(Object obj);   
boolean removeAll(Collection<?> c);  
boolean retainAll(Collection<?> c); remove all except the items of c  
int size();  
Object[] toArray(); // convert the collection into an array  
           // lots of casting ... not recommended  
<T>T[] toArray(T array[]);   // better.  
/\*EX    Arraylist al;     
    String str[] = String[al.size()]  
        str = al.toArray(str);  // ArrayList -> array \*/  
  
}  
2) List i/f  
 : element can be accessed or inserted by the index no.  
  
interface List<E>  extends Collection implements Iterable  
{  
    void add(int index, E obj);  // insert + shift  
    boolean addAll(int index, Collection<? extends E> c);   
    E get(int index);  
    int indexOf(object obj);  
    int lastIndexOf(object obj);  
    ListIterator<E> listIterator();  
    ListIterator<E> listIterator(int index);  
    E remove(int index);  
    E set(int index, E obj);  // replace  
    List<E> subList(int begin, int end) // begin ~  end-1  
  
  
\*\* any class that implements Iterable interface can use  
        for (E x: T)  
    for (ArrayList<String> x : al )  
               println(x);  
  
3) Set  
  
interface Set<E> extends Collection {// no new methods}  
 // all the element should be unique  
 // repetition will be removed  
  
\*\*  Implementing class vs adapter  
  (1) Implementing : should implement all the methods  
            (unless otherwise the class is abstract or interface)  
  (2) class MyAdapterClass implements Collection  
      : you are free to create empty methods (some/all)  
      : heavily used in graphics  
  
  
4) SortedSet  
  
interface SortedSet<E> extends Set   
{  
    Comparator<? super E> comparator(); // factory method ex. size,color,length,...  
    E first();  // default: smallest  
    E last(); // default: largest  
    SortedSet<E> headSet(E end); // all the elements < end  
    SortedSet<E> subSet(E begin, E end); // begin ~ end-1  
    SortedSet<E> tailSet(E begin);// all the elements >= begin  
}      
  
5) NavigableSet  
  : retrieves elements based on the closed value of ...??  
  
interface NavigableSet<E> extends SortedSet  
{  
    E ceiling(E obj);  // search for the smallest element >= obj  
    Iterator<E> descendingIterator();  
    NavigableSet<E> descendingSet();  
    E floor(E obj);  // search for the largest element <= obj  
    NavigableSet<E> headSet(E upperbound, boolean include);  
           // subset less than upperbound. (include upperbound or not include)  
    NavigableSet<E> tailSet(E lowerbound, boolean include);  
    E higher(E obj);   smallest element > obj  
    E lower(E obj);   largest element < obj  
    E pollFirst();  =pop()  
    E pollLast(); =pop\_end()  
    NavigableSet<E> subSet(E lowerbound, boolean low\_include,  
                              E upperbound, boolean upper\_include)  
}