Assert

Boxing / Un-Boxing / Auto-Boxing

Check Java Version in Unix

Class class

Generics

Generic Class

Generic Class (Multiple Types)

Factory Methods

Javac

Mainless Program

Methods

New in Java 7

PrintWriter

Primitive vs Reference

ToolKit Class

**Q: How to check java version?**

$ java –version

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

**Q: What is new in Java 7?**

1) write into file

try

{

fos = new FileOutputStream(“xyz.txt”);

dos = new DataOutputStream(fos);

dos.writeUTF(“hello”);

}

catch(IOException e) { … }

finally

{

try

{

fos.close();

dos.close():

}

catch(IOException e) { ….. }

}

}

//In Java 7 we can do this:

try(FileOutputStream fos = new FileOutputStream (“xyz.txt”);

DataOutputStream dos = new DataOutputStream(fos))

{

dos.writeUTF(“hello”);

}

catch(IOException e) { … }

NOW: automatically closes file(s) / channels

2)

int num = 1\_000\_000; //underscore to separate part of your integers

int num = 0xb000\_11\_00\_11; //b for binary

3)

switch can now use Strings!

String name = “John”;

switch (name) = “John”

{

case “John” : println(“is dude”);

break;

case “Susan” : println(“is chick”);

break; default : println(“none”); }

4.

Combine all exceptions together into one catch:

try

{

…

}

catch(ExceptionOne e) { … };

catch(ExceptionTwo e) { … };

catch(ExceptionThree e) { … };

NOW:

try  
{

….

}

catch(Exception e | Exception k | Exception l);

you can separate exception with | or with ,

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

**Q: How to shorten code? METHODS**

A: create methods.

1. print method

public static void println(Object o)  
{

System.out.println(“” +o);

}

2. die method

public static void die(String … msg)

{

if(msg.length> 0)

{

System.err.println(msg[0]);

}

System.exit(1);

}

3. ContinueY/N Method

public static Boolean askYN()

{

Scanner sc = new Scanner(System.in);

System.out.print(“Continue(Y/N)?”);

while(true)

{

char ans = Character.toUpperCase(sc.nextLine().charAt(0));

if(ans == ‘Y’ || ans == ‘N’) return (ans==’Y’)

System.err.println(“Error: Y or N only”);

}

}

4. RandomNumber Generator

//create method

a <= r <= b

//inclusive

public static int rand(int a, int b)

{

return ( (int)(b-a +1)\* Math.random() +a));

}

Example:

for (int = 0; i < 5; i++) print(rand(3,9));

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

**Q: How to use created methods?**

1. Create method

2. compile using javac MyUtil.java

3. import OR do extends MyUtil

4. create an object of that class and use its (if static methods)

5. extend the class to make it a child of MyUtil

6. Use static methods: MyUtil.println(“ lla”);

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

**Q: Mainless Program**

public class Mainless

{

static

{

System.out.println(“hello mainless”);

System.exit(0);

}

}

Why does this work?

JVM, before looking for main, looks for static blocks and produces them BEFORE MAIN.

Good for fast code.

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

**A: How to data into file?**

printWriter

println(“ “ +

“ “);

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

**Simple Class Box**

public class Box

{

private int w, h, l;

public Box(int wi, int hi, int le)

{

w = wi; h = hi; l = le;

}

public Box(int side)

{

w = side;

h =side;

l=side;

}

OR

this(side, side, side);

//default

public Box()

{

this(0);

}

public int volume();

{

return (l x w x h);

}

public String toString()

{

return (“w = “ + w + “height = …”

}

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

public static String[] loadFile(String filename, int… numblines) throws FileNotFound Exception

{

String nullArray[]= ….;

//**Q: when you want to read from a file ask: does file exist? is file readable?**

//create file object

File f = new File(filename);

//check if file exists

if(!f.exists())

{

System.err.println(filename + “Does not exist”);

System.exit(1);

}

//check if readable

if(!f.canRead())

{

return (nullArray);

}

//next create Scanner

Scanner sc = new Scanner(f);

while(sc.nextLine())

{

//add into elements of array

fileLines.add(sc.nextLine());

}

//check if you can write to the file

//open file

if(!f.canWrite())

{

exit or return null

}

//use printWriter

PrintWriter pw = new PrintWriter(f); //note: will overwrite file completely

//how to append file?

PrintWriter pw = new PrintWriter(new FileWriter(f, true));

//says to call FileWriter Class and append the file

true = append

false = overwrite

PrintWriter Methods:

print

println

printf

/\*\*  
 \* @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();  
  }  
}

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

**Calendar Class / Trace / Factory Methods**

Calendar c = Calendar.getInstance();

Q: Why not Calendar c = new Calendar(); ??

A: Some classes do not have a useful constructor; they create a static Factory Method

When we create a class we don’t know who will use it so we make class flexible.

Example: TRACE

Factor Method = Static Method that returns an Object

public interface Trace //object of trace can be put into file or screen

{

public void setDebug(Boolean debug); //on off

public void debug(String msg); //put msg into screen or file

public void error(String msg); //put error into file/screen

}

import java.io.\*;

public class FileTrace implements Trace

{

private PrintWriter pw;

private Boolean debug;

//create constructor

public FileTrace() throws IOException

{

pw= new PrintWriter( new FileWriter(“trace.log”) );

}

public void setDebug(Boolean d)

{ debug = d; }

public void setdebug(String msg)

{

if(debug)

{

pw.println(“DEBUG” + msg);

pw.flush();

}

}

public void error(String msg)

{

//does not need to check for debug

//always into file

pw.println(“ERROR: “ + msg);

pw.flush();

}

}

//IMPLEMENT TRACE

public class SystemTrace implements Trace //to screen

{

public boolean debug;

public void setDebug(boolean d) {debug = d;}

public void debug(String msg)

{

if(debug)

{

println(“DEBUG” + msg);

}

}

public void error(String msg)

{

System.err.println(“Error:” + msg);

}

}

**FACTORY METHOD**

public class TraceFactory

{

public static Trace getTrace()

{

return SystemTrace(); //FileTrace if want

}

}

Trace t = new TraceFactory.getTrace();

t.error(“bad variable”);

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

**Q: How to read a file from web (url?)**

import java.io.\*; // open stream

import java.util.\*; //scanner

URL url = new URL(“http:// …”);

Scanner sc = new Scanner(ur.openStream());

while(sc.hasNext())

{

..

}

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

**ToolKit Notes:**

Toolkit tk = ToolKit.getDefaultToolKit(); //factory method

Dimension d = tk.getScreenSize(); //size of comp screen

print(d.getWidth() + “ “ d.getHeight());

tk.beep(); //use to make a beep sound

boolean areExtraMouseButtonsEnabled(); // all available on system put into array

FontMatrics getFontMetrics(Font f) // info on Font: size, pixils, width, height

int getScreenResolution(); //pixel per inch

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

**Assert**

-pseudo method used to check if our code is working properly or not

Example:

we have a method --> int getNum() //returns 0 or + int

use assertion to ensure that method is working properly

Two Types:

1)

assert boolean-expression

evaluated:

- if true - nothing happens

- if false - error message

2)

assert boolean-expression : some\_value

if true - nothing

if false - error msg + some\_value

example:

assert result > 0

assert result > 0 : result;

assert(num >= 0 && num <= 10) : "bad number" + num;

default -- asserts are ignored

Q: How to enable assert?

java -ea Doit

java -enableassertions Doit

Q: How to disable?

java -da Doit

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

Note:

System.out // screen of terminal

System.in // for input --> read from terminal

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

JAVAC

javac -d .  xyz.java //compile, create class, move to directory specified by package

--will create a directory if a one does not yet exist

javac -s dirname xyz.java //moves source file into dirname after compilation

--Must create dir first

javac -O Xyz.java // will create an optimizer that will make our code run faster

javac -logfile xyz.java // puts all error msgs into log file (System.err.println() )

Note- if using log.xml then an xml file will be created

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

DATA TYPES

Two types of data types

1. primitive data type ( 8 )

2. reference data type

byte, short, int, long, float, double, boolean, char

Q: Why both: Refernce and primitive?

Answer: So we can pass a refernce to a method

Boxing - primitive to Reference

Un-Boxing - Reference to primitive

-Often we need to convert between primitive and reference

-In java references are sub-types of an Object

-we can assign null to any reference

ie. Integer in  = new Integer();

in = null;

-we cannot assign null to a primitive! //error

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

INT vs INTEGER | BOX – UNBOX

public static Integer sumInteger(List<Integer> ints)

{

Integer sum = 0; //first boxing

for(Integer n : ints ) sum +n; //convert to primitive then convert back to Object

return (sum);

}

-java does not do arithmetic on Reference Data

-we constantly must Box and Unbox

Q: We have the following lines

 List<Integer> bigNumbers = Arrays.asList(100, 200, 300);

 assert sumInteger(bigs) == sum(bigs); //system will un-box and compare, int == int

 assert sumInteger(bigs) == sumInteger(bigs); //not okay, system will compare different memory addresses

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

GENERICS

<data types>  //only reference data types allowed

List<Integer> ints = new ArrayList<Integer>();

Note:

In past if we wanted to put into into List then we had to convert it to Integer

Today autoboxing does this conversion for us.

List<Integer> ints = new ArrayList<Integer>();

ints.add(200); //system autoboxes

//ints.add(new Integer(200)) //old way

//get number

int n = ints.get(0);

//int n = its.get(0).intValue(); //old way that converts Integer to int

public static int sum(List<Integer> ints) //integer version would be 60% slower because of boxing and unboxing

{

  int sum = 0;

  fot( int n : ints ) sum +=n;

  return sum;

}

 List<Integer> ints = Arrays.asLists(1,2,3);

 int s = 0;

 for (int n : ints) s+=n;

 Q: assert s == 6  //ok

 Q: re-write for old version of Java

 for(Integer<Integer> it = ints.iterator(); iterator.hasNext();)

 {

    int n = it.next();

s +=n;

 }

 Interface Iterable<E>

 NOTE: all items in collections implement Iterator<E> so we can use for-each loop

 for-each loop does not implement remove(); so you can't change arraylist structure

 If your class implements this interface then that object can be used in a modern for-loop

 Q:What is insdie Iterable<E>

 interface Iterable<E>

 {

   public Iterator<E> iterator();  //references another interface

 }

 Q: What is inside Iterator<E>

 interface Iterator<E>

 {

   public boolean hasNext();

   public E next();

   public void remove();

  }

  // all methods must be implemented by class for object to be used in for-loop

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

GENERICS "Sub-Type"

one type is a subtype of another if they are related by extends or implements clause

Example:

Integer is a subtype of Number  //Integer extends Number

Double is a subtype of Number

ArrayList<E> is a subtype of List<E>  //List<E> is an interface

Note: Interface can extend or implement an interface?

Collection<E> is a subtype of Iterable<E>

\*\*Subtyping is transitive: a subtype of b subtype of c then a subtype of c

Symetic: if a is subtype of b then b is subtype of a?????

Q: What is Generics?

parametrized types

accept any data type

Q: Why invent Generics?

Allows program to be sort-of typless

Typeless - ruby, pearl - variable x can be used for anything

Typed - java, c++ - user must specify data type for variables/functions

Generic Class - accept any type of generic data

public class Gen<T>

{

  T ob;

  public Gen(T o)

  {

    ob = o;

  }

  public T getob()

  {

    return ob;

  }

  public void showType()

  {

    println("tyle of t " + ob.getClass().getName() );

  }

Class class

NOTE:

Class in java called Class --> any class is an object

getClass() returns object; getName() returns name of class

String x = new String();

x.getClass() = returns String

class DemoGen

{

Gen<Integer> g; //T is now an Integer

g = new Gen<Integer>(88); //initialize g by declaring variable & put data into it

//new in Java 7

//g = new Gen<>(88);

g.showType(); // is Integer

int v = g.getob(); // v = 88 int (unboxing)

Gen<String> sg = new Gen<>(“hello”);

//create Restrictions 🡪 limit class b/c avg() will not work on String

String str = sg.getob();

}

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

Class w/ multiple Generic Types (34)

class TwoGen<T, V>

{

T obj1;

V obj2;

TwoGen(T o1, V o2)

{

obj1 = o1;

obj2=o2;

}

}

Problems using TwoGen