

CIS 27P

Java for Programmers

Class 2

Primitive-Simple Types

- Integers
 - “long”: 64 bits, +/- 9.223×10^{18}
 - “int”: 32 bits, +/- 2.147×10^9
 - “short”: 16 bits, -32,768 to 32,767
 - “byte”: 8 bits, -128 to 127
- Floating point
 - “double”, 64 bits, $1.7e^{-308}$ to $1.7e^{+308}$
 - “float”, 32 bits, $3.4e^{-38}$ to $3.4e^{38}$

Primitive-Simple (cont)

- Characters
 - “char”
 - 16 bits, 0 to 65,536, but displayed as char
 - Unicode -
<http://www.unicode.org/charts/>
 - supports double byte characters
- Booleans
 - “boolean”
 - displayed as “true” or “false”

Literals

- What is a literal?
- Integer literals
 - 324 converts to “int” type
 - force to long if needed:
9928572934572935925L
 - start with 0 to specify octal:
020 == 16
09 will give a compiler error
 - start with 0x to specify hex:
0X20 == 32
0xF == 16
 - if literal is not too large, it’s ok to assign automatic ints to smaller types

Literals (cont.)

- Floating point literals
 - Two possible ways to format
 - 3.14159
 - 1.3E1
 - Automatically converts to “double”
 - Append “f” if float desired:
3.14159F
- Boolean literals
 - “true” or “false” for assignment and display
 - 1 or 0 does not work

Literals (cont.)

- Character literals
 - use single quotes if possible:
‘a’, ‘c’
 - ‘\ddd’ to specify with octal
 - ‘\uxxxx’ to specify with hex
 - escape special characters with ‘\
\’, \”, \\
 - \r = carriage return
 - \n = newline
 - \f = form feed
 - \t = tab
 - \b = backspace

Variables

- Declarations
- Can be declared within any block and has scope for that block
- Can not be used before a value has been given to it
- Can not be re-declared in inner block
- Final variables

Types & Casting

- Java automatically casts when:
 - **two types are compatible**AND
 - **destination is larger**OR
 - **converting from a literal int or double**
- Otherwise, use the cast operator:

```
int c = (int) 5.62;
```
- Automatic type promotion into double and int inside expressions

```
byte b = 50;  
b = (byte) ( b * 2 );
```

```

/**
Author: Java Programmer
Date : June 18, 2001
Title : Type Casting Demo
*/
class Main
{
    public static void main(String args[])
    {
        int i = 15;
        double d = 1.7;
        //i = d; Explicit cast needed
        d = i;      // no problem here
        //cast the parameters:
        System.out.println ("d= " + d);
        //i = 34.7; Explicit cast needed
        i = (int) 34.7;
        System.out.println ("i= " + i);
        char ch = 'A';
        i = ch;
        System.out.println ("i= " + i);
        System.out.println ("ch= " + ch);
        i = 66;
        //ch = i; Explicit cast needed
        ch = (char) i;
        System.out.println ("i= " + i);
        System.out.println ("ch= " + ch);
    }
}

```

Arrays

- **A collection of values of the same type**
- **Various ways to declare and initialize:**

```
int intArray[];  
intArray = new int[5];  
intArray[4] = 7;
```

```
int [] intArray;  
intArray = new int[5];  
intArray[0] = 4*3;  
System.out.println(intArray[0]);
```

```
int intArray[] = new int[5];  
// WRONG intArray[5]=0;
```

```
int intArray[] = { 1, 2, 3, 4, 5};
```

Arrays (cont.)

- To get the size of an array:
 - `intArray.length`
- To copy an array:
 - `System.arraycopy();`
 - Parameters:
 - from
 - position
 - to
 - position
 - length

Multi Dimensional Arrays

- Corresponds to an array of arrays:

```
long lArray[] [] = new long[3][8];  
lArray[2][6] = 5;
```

```
byte bArray[] [] = {{ 1, 2 }, { 3, 4 }};
```

```
System.out.println(bArray[0][1]);
```

Array Sample

```
class ArrayDisplay
{
    public static void main( String [] args )
    {
        int iArray[][] = { {1,2,3}, {4,5,6}, {7,8,9} };
        for ( int c1 = 0; c1 < 3; c1++ )
        {
            for ( int c2 = 0; c2 < 3; c2++ )
            {
                System.out.print("iArray["+c1+""]["+c2+"]:");
                System.out.println(iArray[c1][c2]);
            }
        }
    }
}
```

OUTPUT:

```
iArray[0][0]:1
iArray[0][1]:2
iArray[0][2]:3
iArray[1][0]:4
iArray[1][1]:5
iArray[1][2]:6
iArray[2][0]:7
iArray[2][1]:8
iArray[2][2]:9
```

Exercise

Write the code necessary to sort the following array of integers from the smallest to the largest value and print out the elements in sorted order.

```
int unsortedArray [] = {4, 5, 3, 7, 3, 6, 9};
```

One algorithm you can use is the bubble sort, described with the following pseudo-code:

Where X is the total number of elements:

```
OUTER LOOP A from 1 to (X-1)
  INNER LOOP B from (A+1) to (X)
    if VALUE(A) > VALUE(B)
      switch A & B
    END INNER LOOP
  END OUTER LOOP
```

Wrapper Objects

- Long, Integer, Short, Byte, Double, Float.
- Wraps corresponding primitive types.
- Part of java.lang package.
- Constructors that take a string or a primitive type.
- Access to primitive types with “.byteValue()”
- Use “.equals()” method, comparisons will fail.

Wrapper Objects (cont.)

- Useful for collections classes.
- `MAX_VALUE` and `MIN_VALUE` constants.
- `compareTo()` method.
- `toString()` method.

Strings

- Java class
- Not an array of characters
- Method “.length()”
- Usage
- Array of Strings

Baffling String Test

```
public class TrivialApplication
{
public static void main(String args[])
{
    String a = "hello";
    String b = "hello";
    if ( a==b) System.out.println( "a and b are equal" );
    else System.out.println("a and b are not equal");

    String c = new String("hello");
    String d = new String ("hello");
    if ( c==d) System.out.println( "c and d are equal" );
    else System.out.println("c and d are not equal");

    c = d;
    if ( c==d) System.out.println( "c and d are equal" );
    else System.out.println("c and d are not equal");
}
}
```

Command Line Arguments

- “main” method accepts an array of String objects

```
import java.util.*;

public class ShowArguments
{
    public static void main (String args[])
    {
        for ( int c=0; c < args.length; c++ )
        {
            System.out.println
                ("Argument "+c+" is "+args[c]);
        }
    }
}
```

```
java ShowArguments argument1 argument2 ...
```