

CS1007 lecture #7 notes

tue 12 feb 2002

- news
- command line arguments
- String class
- wrapper classes

news.

- you should be getting back homework#1 this week in recitation
- homework #2 is due thursday
- problems with slides (hopefully!) fixed
- error in last class:
 - the *incorrect* line was:
`inc = Float.valueOf(args[1]).floatValue();`
 - the *corrected* version is:
`inc = (Float.valueOf(args[1])).floatValue();`
(note placement of parenthesis)
- corrected code went on the web page Fri
- I don't always assign reading!

command line arguments (1).

- remember our model of a computer program from the 2nd lecture:
input → CPU → *output*
- there are many ways to get input into your program:
 - command line arguments (introduced last class)
 - * when program starts up
 - * from user, via keyboard
 - * homework #2
 - prompted input (next week)
 - * during program execution
 - * from user, via keyboard
 - * homework #3
 - file input (later in term)
 - * during program execution
 - * from a file
 - * homework #6 (probably)

command line arguments (2).

- when a java program is invoked from the UNIX command line, any values after the program name are *passed into the program*, for use when the program is running.
- the `args` argument to `main` gives you access to these values, for free (i.e., you don't have to do anything special to get them), through the line of code that looks like this:

```
public static void main( String[] args )
```
- you can see how many arguments were passed into the program by using the value of `args . length`
- you can see what the values of the arguments are by looking them up in the `args` list, using an *index*, i.e., a number which indicates which entry in the list you are referring to
- remember that in computer science, we start counting with 0
- so the first value in the argument list is referenced as `args [0]`, and so on

command line arguments (3).

- given the command line:
`unix$ java ex60 A 12 DOG`
then
`args.length` would be equal to 3, and
`args` would look like this:
`arg[0]` "A"
`arg[3]` "12"
`arg[4]` "DOG"
- these are all `Strings`!!

classes.

- a `String` is a *wrapper class*
- every object in Java is a *class*, built up from the primitive data types
- like programs, classes have a “parts” portion and an “instructions” portion
- the “parts” are variables and constants, made up of primitive data types and other classes
- the “instructions” are called *methods*
- we’ve already used one method: `main()`
- usually, you name a method yourself; `main()` is the only exception
- we’ll spend the rest of the term learning about classes that are defined as part of Java (called “native”) and learning how to write our own classes
- today we’ll start with wrapper classes, which are native

wrapper classes.

- every primitive data type has a corresponding wrapper class:

primitive data type	wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character
char []	String

- mostly these are used to convert between `String` and the primitive data types
- `boolean` is handled a bit differently from the numeric data types (details ahead...)
- `char` and `char []` are also handled differently from the other data types (details ahead...)

example.

- here's an example going from `String` → `Integer` → `int`

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> Integer -> int  
        String str = "1234";  
        Integer I  = Integer.valueOf( str );  
        int      i  = I.intValue();  
        System.out.println( "str="+str+" ] int="+i );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```


String → Integer → int.

- note that the two conversion lines:

```
Integer I = Integer.valueOf( str );  
int i = I.intValue();
```

- can be combined in one line as:

```
int i = (Integer.valueOf( str )).intValue();
```

- another (simpler) alternative is:

```
int i = Integer.parseInt( str );
```

- but you should know that no matter which form you use, under the hood, so to speak, the `String` is getting to `int` through the `Integer` wrapper class

converting from String to numeric primitive data types.

- there is a nice symmetry for converting from `String` to all numeric primitive data types
- they all look like the `int` conversion on the previous slides:

```
byte  b = (Byte.parseByte( str )).byteValue();
short s = (Short.parseShort( str )).shortValue();
int   i = (Integer.valueOf( str )).intValue();
long  n = (Long.parseLong( str )).longValue();
float  f = (Float.parseFloat( str )).floatValue();
double d = (Double.parseDouble( str )).doubleValue();
```

- or

```
byte  b = Byte.parseByte( str );
short s = Short.parseShort( str );
int   i = Integer.parseInt( str );
long  n = Long.parseLong( str );
float  f = Float.parseFloat( str );
double d = Double.parseDouble( str );
```

String → Byte → byte

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> Byte -> byte  
        String str = "12";  
        Byte B = Byte.valueOf( str );  
        byte b = B.byteValue();  
        System.out.println( "str=["+str+"] byte="+b );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```

String → Short → short

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> Short -> short  
        String str = "123";  
        Short  S   = Short.valueOf( str );  
        short  s   = S.shortValue();  
        System.out.println( "str=["+str+"] short="+s );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```

String → Long → long

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> Long -> long  
        String str = "12345";  
        long    N    = Long.valueOf( str );  
        long    n    = N.longValue();  
        System.out.println( "str=["+str+"] long="+n );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```

String → Float → float

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> Float -> float  
        String str = "123.45";  
        Float F = Float.valueOf( str );  
        float f = F.floatValue();  
        System.out.println( "str=["+str+"] float="+f );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```

String → Double → double

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> Double -> double  
        String str = "1234.5";  
        Double D  = Double.valueOf( str );  
        double d   = D.doubleValue();  
        System.out.println( "str=["+str+"] double="+d );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```

converting from String to boolean primitive data type.

- `boolean` is handled the same way as `int` in the long form:

```
int    i = (Integer.valueOf( str )).intValue();  
boolean l = (Boolean.valueOf( str )).booleanValue();
```

- but has a different short form:

```
boolean l = Boolean.getBoolean( str );
```

instead of:

```
int    i = Integer.parseInt( str );
```


String → Boolean → boolean

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> Boolean -> boolean  
        String str = "true";  
        Boolean l   = Boolean.valueOf( str );  
        boolean l   = l.booleanValue();  
        System.out.println( "str=["+str+"] boolean="+l );  
        str = "false";  
        l = (Boolean.valueOf( str )).booleanValue();  
        System.out.println( "str=["+str+"] boolean="+l );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```

converting from String to char primitive data type.

- char is handled differently all around
- whereas all the other conversions go from `String` → *wrapper class* → *primitive data type*, there is no way to go directly from `String` to `Character`
- instead you go directly from `String` to char:

```
char c = str.charAt( 0 );
```

- and if you want to get a `Character`, you have to get there from char, so you end up going `String` → char → `Character`

```
Character C = new Character( str.charAt( 0 ) );
```

String → char

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> char  
        String    str = "A";  
        char      c   = str.charAt( 0 );  
        System.out.println( "str=["+str+"] char="+c );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```

String → char → Character → char

```
public class ex7 {  
    public static void main( String[] args ) {  
        // String -> char -> Character -> char  
        String    str = "B";  
        char      c   = str.charAt( 0 );  
        Character C   = new Character( c );  
        char      cc  = C.charValue();  
        System.out.println( "str="+str+" ] char="+cc );  
        System.exit( 0 );  
    } // end of main()  
} // end of class ex7
```