Lesson 57: Number↔`String` Conversions (W19D3)

Balboa High School

Michael Ferraro

January 6, 2016

Do Now

Download a tester class for PS #10's book questions from here and import into your `PS10` project.

Aim

Students will learn how to convert back and forth between numbers (`ints` and `doubles`) and `Strings`.

Number→`String` Conversion

Method #1: Wrapper Classes

- You can "wrap" a primitive `int` or `double` with the corresponding class in Java. Then, simply call the `toString()` method.

Number→`String` Conversion

Method #1: Wrapper Classes

- You can "wrap" a primitive `int` or `double` with the corresponding class in Java. Then, simply call the `toString()` method.
- Ex1:

```
int p = -3;
Integer myInt1 = new Integer( p ); //wrapper
String myIntAsStr1 = myInt1.toString();
// myIntAsStr1 = "-3"
```

Number→String Conversion

Method #1: Wrapper Classes

- You can "wrap" a primitive `int` or `double` with the corresponding class in Java. Then, simply call the `toString()` method.
- Ex1:

```
int p = -3;
Integer myInt1 = new Integer( p ); //wrapper
String myIntAsStr1 = myInt1.toString();
// myIntAsStr1 = "-3"
```

- Ex2:

```
double q = 3.823;
Double myDbl1 = new Double( q ); //wrapper
String myDblAsStr1 = myDbl1.toString();
// myDblAsStr1 = "3.823"
```

Number→`String` Conversion

Method #2: Public Service Methods (`static` Methods)

- You can utilize so-called "public service" methods from the `Integer` and `Double` classes.

Number→`String` Conversion

Method #2: Public Service Methods (`static` Methods)

- You can utilize so-called "public service" methods from the `Integer` and `Double` classes.
- Ex1:

```
int p = -3;
String intStr = Integer.toString( p );
//toString() is overloaded!
```

Number→String Conversion

Method #2: Public Service Methods (`static` Methods)

- You can utilize so-called "public service" methods from the `Integer` and `Double` classes.
- Ex1:

```
int p = -3;
String intStr = Integer.toString( p );
//toString() is overloaded!
```

- Ex2:

```
double q = 3.823;
String dblStr = Double.toString( q );
```

Number→`String` Conversion

Method #3: Another Public Service Method

- The `String` class also provides a `static` method for conversion.

Number→`String` Conversion

Method #3: Another Public Service Method

- The `String` class also provides a `static` method for conversion.
- Ex1:

```
int p = -3;
String intStr = String.valueOf( p );
```

Number→String Conversion

Method #3: Another Public Service Method

- The `String` class also provides a `static` method for conversion.
- Ex1:

```
int p = -3;
String intStr = String.valueOf( p );
```

- Ex2:

```
double q = 3.823;
String dblStr = String.valueOf( q );
//valueOf() is overloaded:
//  accepts int argument
//  accepts double argument
```

Number→`String` Conversion

Method #4: Concatenation with an Empty String

- Save time with this trick!

Number→`String` Conversion

Method #4: Concatenation with an Empty String

- Save time with this trick!
- Ex1:

```
int p = -3;
String intStr = "" + p;
```

Number→`String` Conversion

Method #4: Concatenation with an Empty String

- Save time with this trick!
- Ex1:

```
int p = -3;
String intStr = "" + p;
```

- Ex2:

```
double q = 3.823;
String dblStr = "" + q;
```

`String`→Number Conversion

Use the public service methods `parse*()` from the relevant number class.

String→Number Conversion

Use the public service methods `parse*()` from the relevant number class.

- Ex1: `String`→`int`

```
int a = Integer.parseInt("-266");
System.out.println( a + 1 );                    // prints -265
```

String→Number Conversion

Use the public service methods `parse*()` from the relevant number class.

- Ex1: `String`→`int`

```
int a = Integer.parseInt("-266");
System.out.println( a + 1 );                    // prints -265
```

- Ex2: `String`→`double`

```
double b = Double.parseDouble("0.1679");
System.out.println( b * 10 );                   // prints 1.679
```

String→Number Conversion

Consider when getting user input via Scanner:

```
import java.util.Scanner;

Scanner keybd = new Scanner(System.in);
System.out.print("Enter an integer: ");
int a = keybd.nextInt();

// breaks when non-int is typed!
```

String→Number Conversion

Consider when getting user input via Scanner:

```
import java.util.Scanner;

Scanner keybd = new Scanner(System.in);
System.out.print("Enter an integer: ");
String str = keybd.nextLine();

int a = Integer.parseInt( str );

// try this -- what happens when a non-int
// is typed by the user?
```

String→Number Conversion

Consider when getting user input via Scanner:

```
import java.util.Scanner;

Scanner keybd = new Scanner(System.in);
System.out.print("Enter an integer: ");
String str = keybd.nextLine();

int a = Integer.parseInt( str );

// try this -- what happens when a non-int
// is typed by the user?
```

See a working example here.

`String` Formatting

Another look at {`int`|`double`}→`String` conversion: When the `String` should have a specific format, use

- `java.text.DecimalFormat` ← outlined on Litvin p220
- `System.out.printf()` ← origins in C, inherited by numerous languages

String Formatting via printf()

`System.out.printf()` takes a variable number of arguments:

```
System.out.printf(
    <String describing format>,
    <num1>,
    <num2>,
    ...
);
```

String Formatting via printf()

Example:

```
int month = 2;
int day = 5;
int year = 2008;
double price = 26.5;

System.out.println( month + "/" + day + "/" + year);
System.out.println( price );

//  2/5/2008
//  26.5
```

Yuck! :(

String Formatting via printf()

Example:

```
int month = 2;
int day = 5;
int year = 2008;
double price = 26.5;

System.out.printf( "%02d/%02d/%d", month, day, year );
System.out.printf( "$%.2f", price );

//  02/05/2008
//  $26.50
```

Better :)

String Formatting via `printf()`

- Download and import `PrintfExamples.java`
- Look at the examples and make sure you understand how to...
  - print `ints` with a specified number of digits (padding the left with zeroes as needed) **(test 1)**
  - print `doubles`/`floats` with
    - a specified number of digits after the decimal point **(test 2)**
    - a specified number of total digits (before and after the decimal point) **(test 3)**

Next. . .

Continue working on the book problems from PS #10, §4.

HW

Work on PS #10, §§5-7, inclusive.