U N I V E R S I T Y

CS 180: Problem Solving and Object-Oriented Programming

Lecture 4: Conversions, Characters, and Strings

Dr. Jeff Turkstra

Lecture 04

- Casting and autoboxing
- Literals and constants
- Characters and strings
- Math
- Sample program

boolean

```
int i = 7;
if (i != 0) {
  …
}
```

…only in C

```
if (i) {
  …
}
```


Conversions

- Identity
- Widening primitive
- Narrowing primitive
- Widening reference
- Narrowing reference
- Boxing
- Unboxing
- Unchecked
- Captures
- String
- Value

Widening

- byte to short, int, long, float, or double
- short to int, long, float, or double
- char to int, long, float, or double
- int to long, float, or double
- long to float or double
- float to double

Wider

- Sometimes called type promotion
- Usually does not lose information
  - float to double can lose magnitude information
  - int or long to float may lose precision
- Never results in a runtime exception

Narrowing

- short to byte or char
- char to byte or short
- int to byte, short, or char
- long to byte, short, char, or int
- float to byte, short, char, int, or long
- double to byte, short, char, int, long, or float

Narrower

- May lose information about overall magnitude as well as precision and range
  - double to float
  - Signed integer to integral type (byte, short, int, long, char)
- Often not permitted without a cast

Casting

- Converts, at runtime, a value of one numeric type to a similar value of another numeric type
  - Except boolean
- Checks, at runtime, that a reference value refers to an object of compatible class
- You are telling the compiler: "I know better than you do what should happen here."

- Upcasting – type promotion, widening
  - Permitted and can happen automatically
- Downcasting – narrowing
  - Dangerous
  - Precision is lost
  - Prevented by default
  - Must be done explicitly

Autoboxing

- Primitive types have corresponding "object wrapper" classes
  int → Integer, double → Double, etc

```
Character myChar = 'a';
```

- Conversion the other way is called unboxing

char

- Remember, everything is a number (0s and 1s)
- 'A' = 65, 'B' = 66, 'a' = 97, etc
- 16-bits
- Building block of Strings

- ASCII subset, 0-127
  - English alphabet, numbers, punctuation, special characters
- Latin-1 extension, 128-255
  - non-English (Romanized) characters and punctuation (dipthongs, accents, circumflexes, etc)
- Unicode 1 extension, 256-65535
  - non-Romanized alphabetic characters from Cyrillic, Hebrew, Chinese, Japanese, etc

char Literals

- Single character surrounded by single quotes
  - 'A', 'a', 'x', '0', '!', ',', '"', '&'
- Escape sequences
  - '\t' tab
  - '\n' newline
  - '\'' single quote
  - '\\' backslash
  - '\uxxxx' hexadecimal xxxx from unicode

Operations

- Autoboxes to Character

  ```
  System.out.println('A');
  ```

  "Hello" + '!' → "Hello!"
- Remember .toString()?
- Treated as integer for arithmetic operations
  - 'a' + 0 → 97
  - 'z' - 'a' → 25

Character methods

isDigit(char)
isLetter(char)
isLetterOrDigit(char)
isLowerCase(char)
isUpperCase(char)
isWhiteSpace(char)
toLowerCase(char)
toUpperCase(char)

instanceof

- Compares an object to a specified type
  - No primitive types
- Can test if an object is an instance of a class, a subclass, or a class that implements a particular interface
- If it can be determined at compile time that none of the above is true, a compile error is emitted instead

Level	Operator	Description	Associativity
16	[ ] . ( )	access array element access object member parentheses	left to right
15	++ --	unary post-increment unary post-decrement	not associative
14	++ -- + - ! ~	unary pre-increment unary pre-decrement unary plus unary minus unary logical NOT unary bitwise NOT	right to left
13	( ) new	cast object creation	right to left
12	* / %	multiplicative	left to right
11	+ -	additive string concatenation	left to right
10	<< >> >>>	shift	left to right
9	< <= > >= instanceof	relational	not associative
8	== !=	equality	left to right
7	&	bitwise AND	left to right
6	^	bitwise XOR	left to right
5	|	bitwise OR	left to right
4	&&	logical AND	left to right
3	||	logical OR	left to right
2	?:	ternary	right to left
1	= += -= &= *= /= %= ^= |= <<= >>= >>>=	assignment	right to left

Pre/post increment

```
int a = 5;
int b = 6;
System.out.println(“a = “ + a++);
System.out.println(“b = “ + ++b);
System.out.println(“a = “ + a);
System.out.println(“b = “ + b);
```


Reference types

- Declarations create space for reference to an object, not the object itself
- Again, must use new to construct the object

Strings (again)

- Built-in class, no need to import
- Literals are objects (“hello”)

```
String greeting = “Hello”;
String greeting = new String(“Hello”);
```

- String variables hold references to objects
- Can also do things like...

```
char[] howdyArray = {‘h’, ‘o’, ‘w’, ‘d’, ‘y’};
String howdyString = new String(howdyArray);
```


Concatenation

```
string1.concat(string2);
”hello, “.concat(“my name is”);
”not “ + “slim shady” + ‘!’;

int a = 7;
System.out.println(“a = “ + a);
```

- Invokes toString() method if it is not a string
  - Remember autoboxing?

Comparing strings

- JVM has a String pool
- It will search for literals in it before creating a new object
  - If it exists, simply updates reference to point to it
- So what does == do?
- Use the .equals() method instead
  - s1.equals(s2)

String to numeric value

- Remember the wrapper classes?

```
Integer.parseInt(“4000”);
Double.parseDouble(“66.23457”)
```

- Watch for NumberFormatException

Min and max

```
Integer.MAX_VALUE
Integer.MIN_VALUE
Double.MAX_VALUE
Double.MIN_VALUE
```

- etc

Math package

- Math.pow(x, y) → x^y
- Math.log(x) → ln(x)
- Math.log10(x) → log(x)
- Math.sqrt(x)
- Math.sin(a) a is in radians
- Math.asin(a)
- Math.toRadians(d)
- Math.exp(x) → e^x

College cost calculator

```
import java.util.Scanner;

public class CollegeCosts {
  public static void main(String[] args) {
    Scanner in;

    // Input variables...
    String firstName, lastName;
    double semesterTuition, monthlyRent, monthlyFood;
    double annualInterest;
    int years;

    // Computed variables...
    double yearlyCost, fourYearCost, monthlyInterest;
    double monthlyPayment, totalLoanCost;
    System.out.printf("Welcome to the College Cost Calculator!\n");
    in = new Scanner(System.in);
```



```
    // Prompt for input values...
    System.out.printf("Enter your first name:       ");
    firstName = in.next();
    System.out.printf("Enter your last name:        ");
    lastName = in.next();
    System.out.printf("Enter tuition per semester:  $");
    semesterTuition = in.nextDouble();
    System.out.printf("Enter rent per month:        $");
    monthlyRent = in.nextDouble();
    System.out.printf("Enter food cost per month:   $");
    monthlyFood = in.nextDouble();
    System.out.printf("Annual interest rate:         ");
    annualInterest = in.nextDouble();
    System.out.printf("Years to pay back your loan:  ");
    years = in.nextInt();

    // Compute results...
    yearlyCost = semesterTuition * 2.0 +
      (monthlyRent + monthlyFood) * 12.0;
    fourYearCost = yearlyCost * 4.0;
    monthlyInterest = annualInterest / 12.0;
    monthlyPayment = fourYearCost * monthlyInterest /
      (1.0 - Math.pow(1.0 + monthlyInterest, -years * 12.0));
    totalLoanCost = monthlyPayment * 12.0 * years;
```



```
    // Print results...
    System.out.printf("\n");
    System.out.printf("College costs for %s %s\n",
      firstName, lastName);
    System.out.printf("*********************************\n");
    System.out.printf("Yearly cost:          $%10.2f\n",
      yearlyCost);
    System.out.printf("Four year cost:       $%10.2f\n",
      fourYearCost);
    System.out.printf("Monthly loan payment: $%10.2f\n",
      monthlyPayment);
    System.out.printf("Total loan cost:      $%10.2f\n",
      totalLoanCost );
  }
}
```


Boiler Up!