Complex Logic, Introduction to Wrapper Class, and Composition

Colorado State University
Department of Computer Science

Slides Originally Created by Marcia Moraes
(marcia.moraes@colostate.edu)

Weekly Announcements!

TODO Reminders:

- Reading 8 (zybooks) – you should have already done that ☺
- Lab 05 – self explanation needs to be done in lab
- Reading 9
- Lab 06
- Reading 10 (zybooks)
- Keep practicing your RPAs in a spaced and mixed manner ☺

Review: Conditionals Operators

Conditional Operators
a `>` b - **true** when a is greater than b
a `<` b - **true** when a is less than b
a `>=` b - **true** when a is greater than *or* equal to b
a `<=` b - **true** when a is less than *or* equal to b
a `==` b - **true** only when a equals b
a `!=` b - **true** only when a does not equal b

Review: Conditionals Operators

When we want to compare two objects (such as Strings), we use `obj1.equals(obj2)`

- Why?
  - `==` only compares the local stack, which is just the memory addres of the object!
  - `.equals` allow the object to decide what that means
    - For `String` that means each character is compared from left to right.
- The .equals() method returns `true` or `false`

How can we combine different conditions?

Use Logical Operators to do Complex Conditional!

Complex Conditionals

a	b	a AND b
False	False	False
False	True	False
True	False	False
True	True	True

All must be true

a	b	a OR b
False	False	False
False	True	True
True	False	True
True	True	True

At least one must be true

a	NOT a
False	True
True	False

Negates the original condition

Logical Operators

- Logical and - **&&**
  - All must be true

```
int puppyCounter = 10;
if(puppyCounter > 5 &&  puppyCounter < 10) {
     System.out.println(“Yay, puppies”);
}
```

- Logical or - **||**
  - Any statement can be true
  - Continues checking even if false

```
String cmd = “exit”;
if(cmd.contains(“x”) || cmd.contains(“X”)) {
     System.out.println(“Goodbye”);
}
```

Precedence rules for arithmetic, logical, and relational operators

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
!	! (logical NOT) is next	`! x || y` is evaluated as `(!x) || y`
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
< <= > >=	Relational operators	`x < 2 || x >= 10` is evaluated as `(x < 2) || (x >= 10)` because < and >= have precedence over ||.
== !=	Equality and inequality operators	`x == 0 && x != 10` is evaluated as `(x == 0) && (x != 10)` because == and != have precedence over &&. == and != have the same precedence and are evaluated left to right.
&&	Logical AND	`x == 5 || y == 10 && z != 10` is evaluated as `(x == 5) || ((y == 10) && (z != 10))` because && has precedence over ||.
||	Logical OR	|| has the lowest precedence of the listed arithmetic, logical, and relational operators.

Logical Operators – In Class Activity - Attendance

- Only compare true or false
  - Every statement on each side must evaluate as true or false to use the logical operation.
- Mark valid or invalid for the statements below and explain your answer:

A) `!((4 + 1) < (2 * 0))`

B) `10/2 > 5 || (3 + 4) < 10 && true`

C) `10 > 5 && 10`

Review

- Conditional Operators
  - Compare primitives
  - Objects need methods
    - equals
    - equalsIgnoreCase
- Logical Operators
  - Compare boolean only – true and false
  - As such, everything must evaluate to true or false before they are used

Wrapper Class

- A *primitive type* variable directly stores the data for that variable type, such as int, double, or char.
- A *reference type* variable can refer to an instance of a class, also known as an object.
- *Wrapper classes* that are built-in reference types that augment the primitive types
  - Meaning that every primitive type has a class associated which provides methods to be used

Wrapper Class

Reference type	Associated primitive type
Character	char
Integer	int
Double	double
Boolean	boolean
Long	long

Example: using Integer.parseInt

```
public class IntegerParseIntExample {
   public static void main(String[] args) {
      int decimalExample = Integer.parseInt("20");
      int signedPositiveExample = Integer.parseInt("+20");
      int signedNegativeExample = Integer.parseInt("-20");

      System.out.println("Value = "+decimalExample);
      System.out.println("Value = "+signedPositiveExample);
      System.out.println("Value = "+signedNegativeExample);
   }
}
```

Composition

- A composition in Java between two objects associated with each other exists when there is a strong relationship between one class and another.
- For example, one class has an attribute that is an object of another class.

Composition: Example

```java
public class Book {
    // Instance variables --> attributes
    private String title;
    private String author;
    // Constructor of this class
    public Book(String title, String author)
    {
        this.title = title;
        this.author = author;
    }
    public String toString(){
        return "Title: " + title + " Author: " + author;
    }
}
```

```java
public class Library {
    private Book book1;
    private Book book2;
    private Book book3;
    public Library(){
        book1 = null;
        book2 = null;
        book3 = null;
    }
    public boolean addBook(Book book){
        if(book1 != null && book2 !=null && book3 != null)
            return false;
        if(book1 == null) book1 = book;
        else if(book2 == null) book2 = book;
        else if(book3 == null) book3 = book;
        return true;
    }
    public String toString(){
        String msg = "";
        if(book1 != null) msg += book1 + "\n";
        if(book2 != null) msg += book2 + "\n";
        if(book3 != null) msg += book3 + "\n";
        if(msg.equals("")) msg = "No books in the library!";
        return msg;
    }
}
```

Instance variables

Constructor

Composition: Example

```
public class AppLibrary {
    public static void main(String args[]){
        Library lib = new Library();
        System.out.println(lib.toString());
        Book b1 = new Book("Death on the Nile", "Agatha Christie");
        if(lib.addBook(b1)) System.out.println("Book added!");
        else System.out.println("No more space in the library!");
        System.out.println(lib.toString());
    }
}
```

Lets take a look on Lab 05

- First step
  - Understand where wrapper class and composition were used
- Second step
  - Understand what the classes provided are doing to be able to implement the method addRow – this is the only method you need to implement
- The TAs will provide additional help tomorrow during your lab time
- https://github.com/CSU-CompSci-CS163-4/Lab05LogicalGarden