Welcome to CSSE 220

- We are excited that you are here:
  - Start your computer and get ready for our first class session.
  - Pick up a quiz from the back table and answer the first two questions.

Course Introduction, Starting with Java

CSSE 220—Object-Oriented Software Development

Rose-Hulman Institute of Technology

Agenda

- Roll Call
- Instructor intro
- A few administrative details
- Verify Eclipse and Subclipse configuration
- Java *vs*. Python and C
- Examine and modify simple Java programs

Daily Quizzes

- I expect you to answer every question.
- Stop me if I don't cover a question!

Q1 – 2

Roll Call, Introductions

- Tell me what you prefer to be called
- For introductions give:
  - Name (nickname)
  - Residence Hall / Floor if on campus
  - Hometown
  - Something you enjoy or are very good at
- Student assistants: introduce yourselves.
- Instructor introduction

Q3

A Tour of the On-line Course Materials

- ANGEL
- Syllabus
- Schedule

Evening lab assistants, F-217

- 7-9 PM Sunday-Thursday

Programming is not a spectator sport

- And neither is this course
- Ask, evaluate, respond, comment!
- Is it better to ask a question and risk revealing your ignorance, or to remain silent and perpetuate your ignorance?

It's OK to interrupt during class discussions

- Even with statements like, *"I have no idea what you were just talking about."*
- We want to be polite, but in this room learning trumps politeness!
- I do not intend for classroom discussions to go over your head. Don't let them!

Introduction to Java

Things Java Has in Common with Python

- Classes and objects
- Lists (but no special language syntax for them like Python)
- Standard ways of doing graphics and GUIs
- A huge library of classes/functions that make many tasks easier
- A nicer Eclipse interface than C has

Things Java Has in Common with C

- Primitive types: **int, char, long, float, double**
- Static typing
- Similar syntax and semantics for **if, for, while, break**, function definitions, …
- Semicolons
- Program execution begins with **main( )**
- Comments: // and /* … */
- Arrays are *homogeneous*, and size must be declared at creation; size cannot change

Why Java?

- Widely used in industry for large projects
  - From cell phones
    - including smart phones—Android platform
  - To global medical records
- Object-oriented (unlike C)
- "Statically type safe" (unlike Python, C, C++)
- Less complex than C++
- Part of a strong foundation
- Most popular language according to the TIOBE Programming Community Index [November 2011]

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

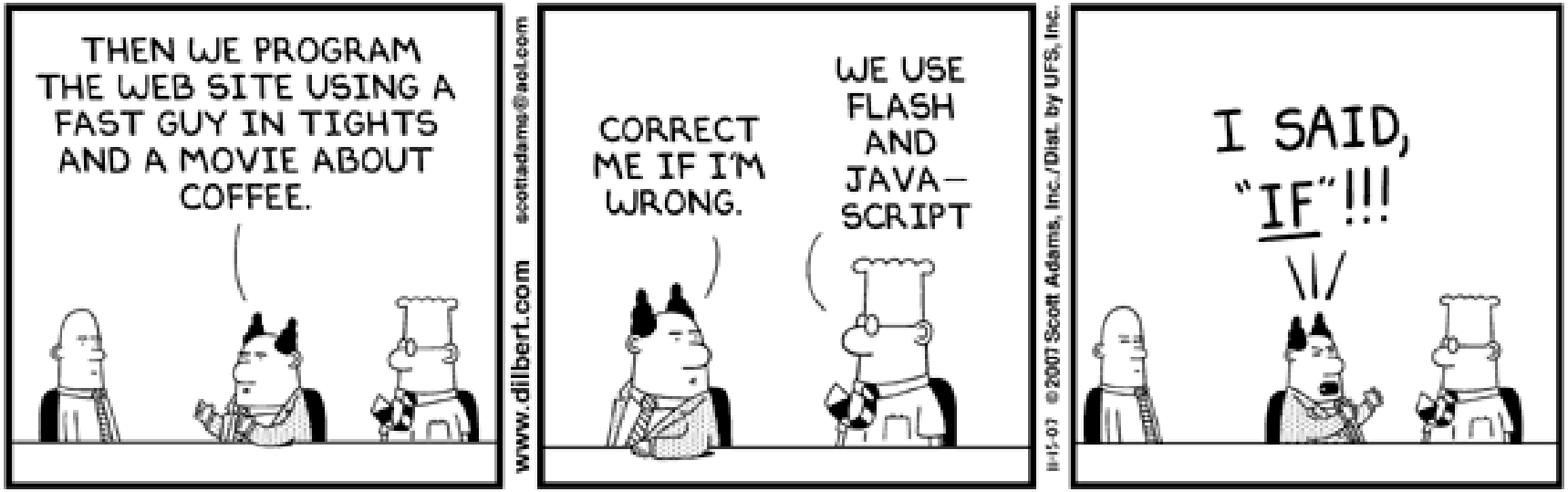
Guess what language is #2

Q8

Java 7 and Eclipse Juno

- I emailed you about upgrading.
- If you haven't done it, you can now:
  - http://www.rose-hulman.edu/class/csse/resources/Eclipse/csse220-update.htm
- Or do it later, which is more work for you.

Interlude: JavaScript and Java have little in common (except their names)



From Wikipedia (edited, bullets added to enhance PowerPoint readability):

- The change of name to JavaScript roughly coincided with Netscape adding support for Java technology in its web browser.
- The name caused confusion, giving the impression that JavaScript was a spin-off of Java.
- The choice has been characterized by many as a marketing ploy by Netscape to give JavaScript the cachet of what was then the hot new web-programming language.
- It has also been claimed that the language's name is the result of a co-marketing deal between Netscape and Sun, in exchange for Netscape bundling Sun's Java runtime with its then-dominant browser.

Checkout today's project (HW1)

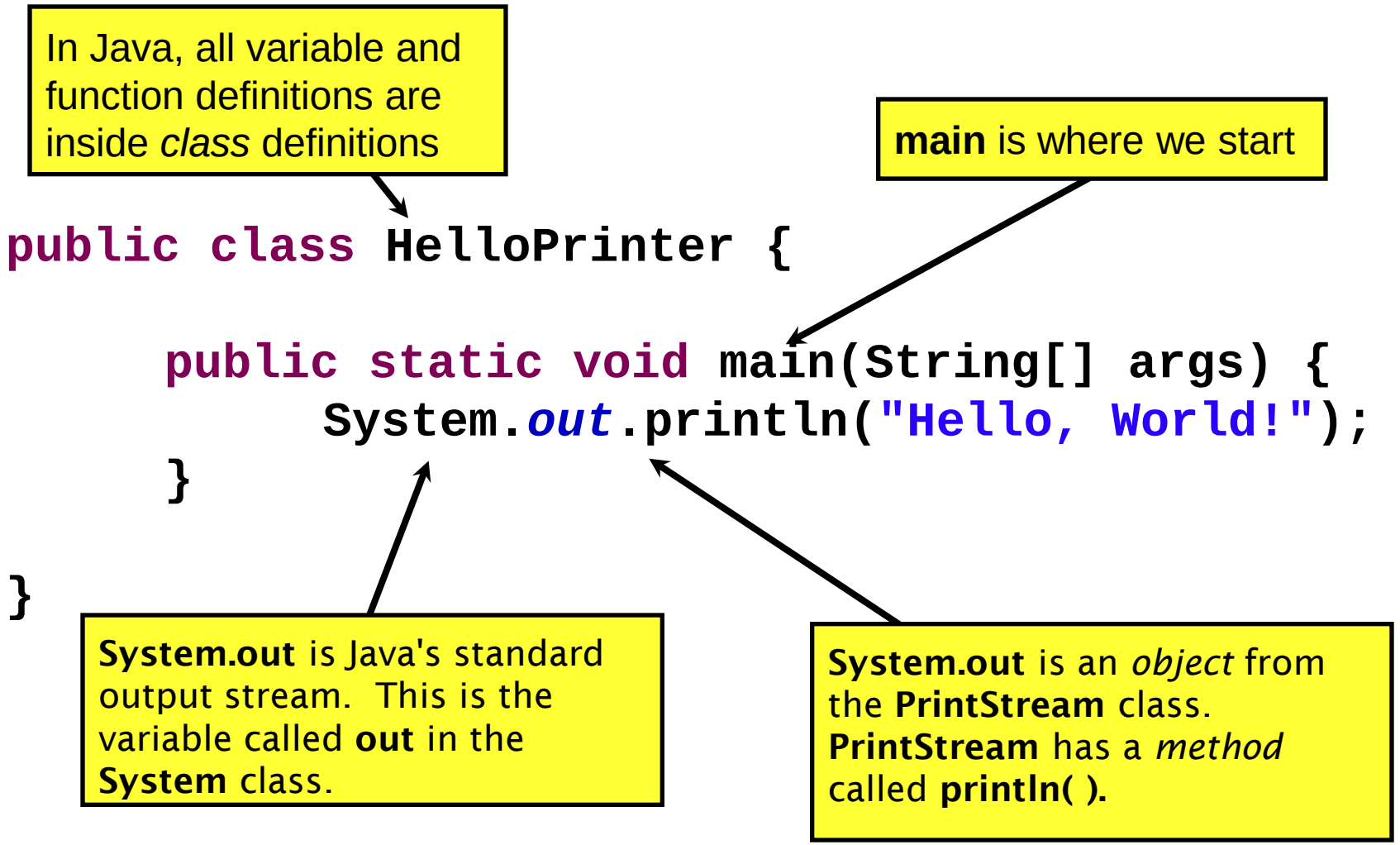
- New Eclipse workspace, Java perspective (there is probably already a csse220 workspace on your computer)
- Go to SVN Repository view, at bottom of the workbench
  - If it is not there, **Window → Show View → Other → SVN → SVN Repositories**
- Right-click in SVN view, then choose **New Repository Location**
  - **http://svn.csse.rose-hulman.edu/repos/csse220-201330-*your_username***
- Right-click **HW1** project and choose **Checkout**
  - **Accept default options**

Get help immediately if you're stuck!

HelloPrinter.java

- To run a Java program:
  - Right-click the .java file in Package Explorer view
  - Choose **Run As → Java Application**
- Change the program to say hello to a person next to you
- Introduce an error in the program
  - See if you can come up with a different error than the person next to you
- Fix the error that the person next to you introduced

A First Java Program



A Second Java Program

```
public class Factorial {
    public static final int MAX = 17;

    public static int factorial(int n) {
        int product;

        product = 1;
        for (int i = 2; i <= n; i++) {
            product = product * i;
        }

        return product;
    }

    public static void main(String[] args) {
        for (int i = 0; i <= Factorial.MAX; i++) {
            System.out.print(i);
            System.out.print("! = ");
            System.out.println(factorial(i));
        }
    }
}
```

Define a constant, MAX

Except for **public static** and the declaration of the loop counter *inside* the ***for*** header, everything about this function definition is identical to C.

println (below) terminates the output line after printing; **print** doesn't.

This ***class*** is called *Factorial*. It has one ***field*** called MAX and two ***methods***: *factorial* and *main*.

Make a new class (**File ~ New ~ Class**) called ***Factorial*** (check the box to let Eclipse type *main* for you). Enter & run the Factorial code. What happens when *i* = 14? Why?

Q10 – 12

```java
/**
 * Has a static method for computing n!
 * (n factorial) and a main method that
 * computes n! for n up to Factorial.MAX.
 *
 * @author Claude Anderson et al.
 */
public class Factorial {
    /**
     * Biggest factorial to compute.
     */
    public static final int MAX = 17;

    /**
     * Computes n! for the given n.
     *
     * @param n
     * @return n! for the given n.
     */
    public static int factorial (int n) {
        ...
    }

    ...
}
```

Javadoc comments

We left out something important on the previous slide – comments!

Java provides Javadoc comments (they begin with /**) for both:

- Internal documentation for when someone reads the code itself
- External documentation for when someone re-uses the code

Comment your own code now, as indicated by this example. Don't forget the @author tag in HelloPrinter.

Writing Javadocs

- Written in special comments: /** … */
- Can come before:
  - Class declarations
  - Field declarations
  - Constructor declarations
  - Method declarations
- Eclipse is your friend!
  - It will generate Javadoc comments automatically
  - It will notice when you start typing a Javadoc comment

In all your code:

- Write appropriate comments:
  - Javadoc comments for public fields and methods.
  - Explanations of anything else that is not obvious.
- Give self-documenting variable and method names:
  - Use name completion in Eclipse, Ctrl-Space, to keep typing cost low and readability high
- Use Ctrl-Shift-F in Eclipse to format your code.
- Take care of all auto-generated TODO's.
  - **Then delete the TODO comment**.
- Correct ALL compiler warnings. Quick Fix is your friend!

Identifiers (Names) in Java

- The rules:
  - Start with letter or underscore (_)
  - Followed by letters, numbers, or underscores

- The conventions:
  - `variableNamesLikeThis`
  - `methodNamesLikeThis(…)`
  - `ClassNamesLikeThis`
- You should follow the conventions!

Q13 – Q15

Variables in Java

- Like C:
  - `int xCoordinate = 10;`

- But Java catches some mistakes:

```
int width, height, area;
area = width * height;
```

  - Java will detect that `width` and `height` aren't initialized!

Using Objects and Methods

- Works just like Python:
  - *object.method(argument, ...)*

"Who does what, with what?"

Implicit argument

Explicit arguments

- Java Example:

```
String name = "Bob Forapples";
PrintStream printer = System.out;

int nameLen = name.length();
printer.printf("'%s' has %d characters", name, nameLen);
```

The dot notation is also used for *fields*

Q16

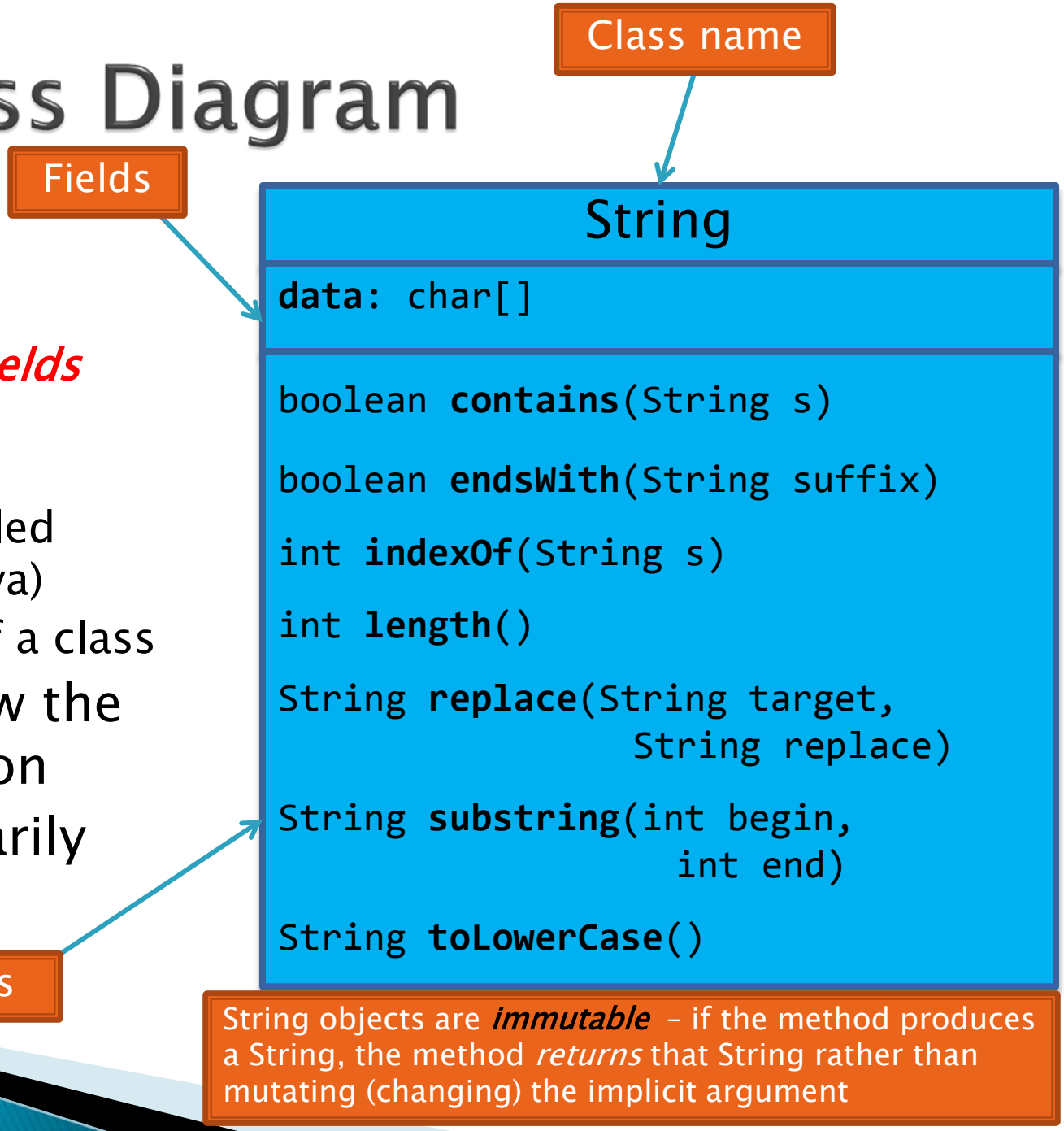
Separating Use from Implementation

- We can use an object's methods without knowing how they are implemented
  - Recall zellegraphics from csse 120:
    `line.setWidth(5)`

UML Class Diagram

- Shows the:
  - *Attributes* (data, called *fields* in Java) and
  - *Operations* (functions, called *methods* in Java)

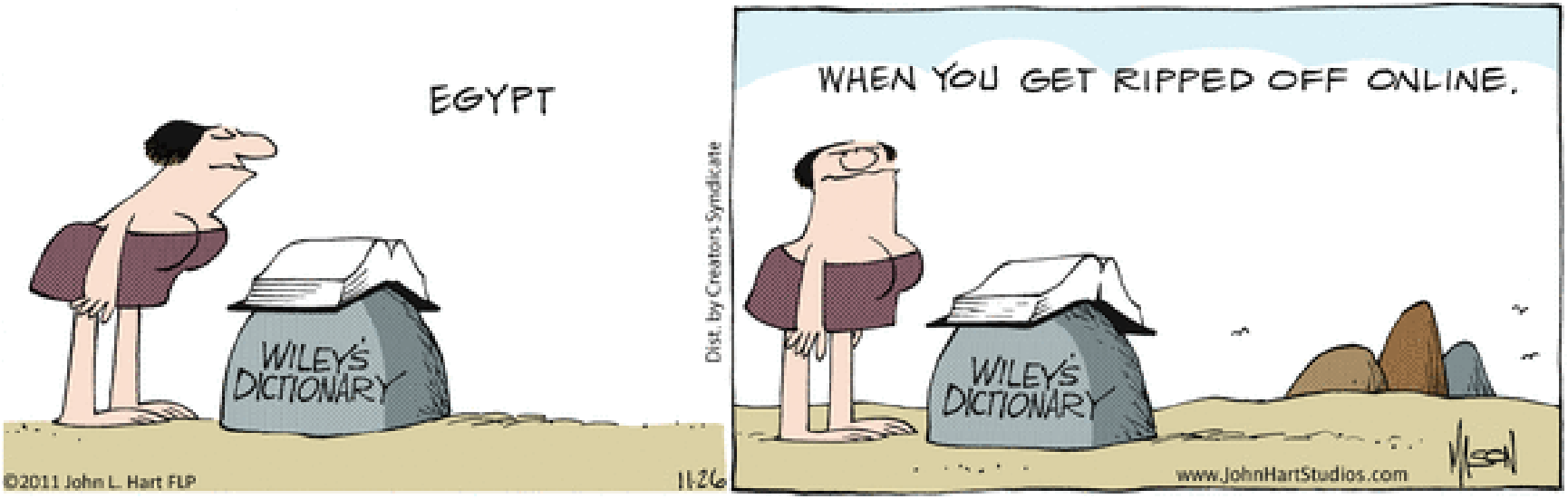
  of the objects of a class
- Does *not* show the implementation
- Is *not* necessarily complete



Exercise

- Checkout ObjectsAndMethods from SVN
  Work on UsingStrings.java

Interlude


EGYPT
WILEY'S DICTIONARY
©2011 John L. Hart FLP
11-26
Dist. by Creators Syndicate
WHEN YOU GET RIPPED OFF ONLINE.
WILEY'S DICTIONARY
www.JohnHartStudios.com

Passing Parameters

- Arguments can be any expression of the “right” type
  - See example…
- What happens if we try to give `substring()` an explicit argument that isn’t a number?
  - How does the compiler know that `rhit.length()` evaluates to a number?
  - What’s the return type of `length()`?

```
String rhit = “Rose-Hulman”;
System.out.println("Rose");
System.out.println(rhit.substring(0, 4));
System.out.println(rhit.substring(0, 2+2));
System.out.println(rhit.substring(0, rhit.length() - 7));
System.out.println("Rose-Hulman".substring(0, 4));
```

Primitive types

Primitive Type	What It Stores	Range
`byte`	8-bit integer	–128 to 127
`short`	16-bit integer	–32,768 to 32,767
`int`	32-bit integer	–2,147,483,648 to 2,147,483,647
`long`	64-bit integer	-2^{63} to $2^{63} - 1$
`float`	32-bit floating-point	6 significant digits (10^{-46}, 10^{38})
`double`	64-bit floating-point	15 significant digits (10^{-324}, 10^{308})
`char`	Unicode character	
`boolean`	Boolean variable	`false` and `true`

figure 1.2

The eight primitive types in Java

Most common number types in Java code

Q17

Exercise

- Work on SomeTypes.java

Constructing Objects

x, y, width, height

- Example:

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

- Several steps are happening here:
  1. Java reserves space for a *Rectangle* object
  2. Rectangle's *constructor* runs, filling in slots in object
  3. Java reserves a variable named *box*
  4. *box* is set to refer to the object

Q18

Accessors and Mutators

- *Accessor* methods
  - Get a value from an object
  - Examples:
    - `box.getHeight()`
    - `box.getWidth()`

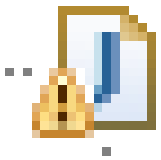
- *Mutator* methods
  - Change the *state* of an object (i.e., the value of one or more fields)
  - Examples:
    - `box.translate(10, 20)`
    - `box.setSize(5, 5)`

Tip: Use mutators with care!

Q19-Q20

Reminder: In all your code:

- Write appropriate comments:
  - Javadoc comments for public fields and methods.
  - Explanations of anything else that is not obvious.
- Give self-documenting variable and method names:
  - Use name completion in Eclipse, Ctrl-Space, to keep typing cost low and readability high.
- Use **Ctrl-Shift-F** in Eclipse to format your code.
- Take care of all auto-generated TODO's.
  - Then delete the TODO comment.
- Correct ALL compiler warnings.
  - Quick Fix is your friend!



Homework Due Before Next Session

- HW1a, linked from the schedule page

HW1b Due Friday by 11:59 PM

Q21-Q22