Object Intro and Miscellaneous

Import *ObjectIntroAndMiscPractice* project from git repo

Writing clean code

Comments are only the last resort

Operations

- Give operations descriptive names
  Specification by naming
- Don’t make functions too long
  Implement each operation do one thing, and one thing well
- Rather than commenting an unclear function, modify the code so it is clear
  e.g., create helper operations that capture a piece of the more complicated

Naming in Java

- Having good names for functions and variables is one of the best things you can do to make your program understandable
- The *camelCase* convention for variables and operations:
  - `variableNamesLikeThis`
  - `methodNamesLikeThis(…)`
- Each word capitalized convention for `ClassNamesLikeThis`
- You should follow the conventions!

```
/**
 * Has a static method for computing n!
 * (n factorial) and a main method that
 * computes n! for n up to Factorial.MAX.
 *
 * @author Mike Hewner & Delvin Defoe
 */
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

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

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:

- See https://github.com/RHIT-CSSE/csse220/blob/master/Docs/grading_guide.md

- Write appropriate comments:
  - Javadoc comments primarily for classes.
  - Explanations of anything else that is not obvious in any spot.
- 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.

Q1

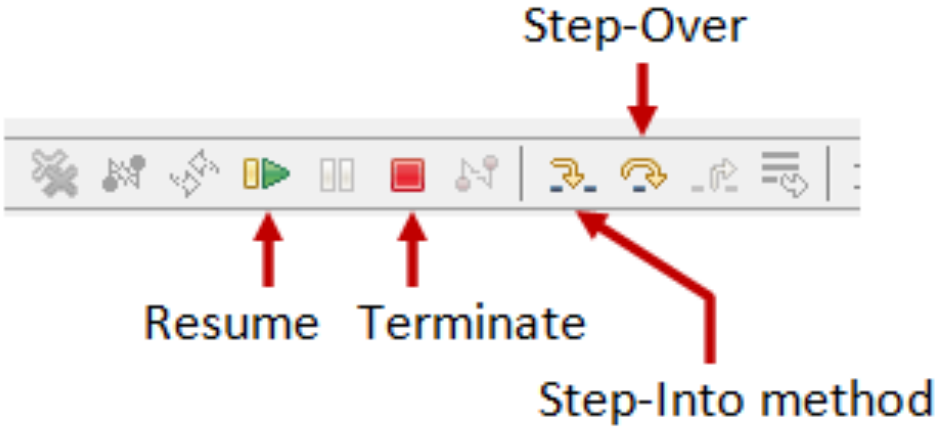
Debugging

Comments That Help with Debugging

- Always use { … } for a block of code in:
  - `if` statements
  - `for` loops
  - `while` loops
- Always mark the end of a block with an: `// end`
  - `if { code here } // end if`
  - `for ( … ) { loop body here } // end for`
  - `while ( … ) { loop body here } // end while`

Debugging—Demo

- Debugging Java programs in Eclipse:
  - Set a breakpoint where you want to start
  - Launch using the bug icon
  - Single stepping: *step over* and *step into*
  - Inspecting variables

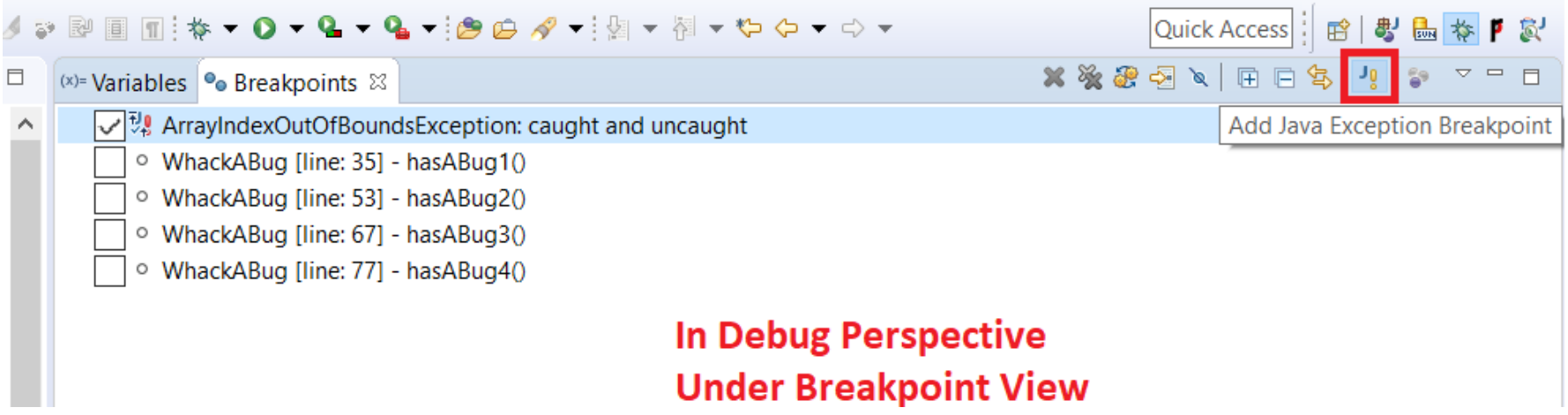


Q2-4

Exception Breakpoint

Very useful when an exception is happening but you don't know where or why

- Exception Tab
- Exclamation point button
- Find the exception type you want
- Add a breakpoint



gotcha: Strings in java are immutable

- No method from the string class will modify the content of string object that is the controlling object
- Example:

```
String s1 = “breakpoint”;
s1.substring(0, 5);
// s1 is the controlling object
```

- All String methods that appear to modify the controlling object, instead return a new string
- Need a catcher object:

```
String s1 = “breakpoint”;
String s2 = s1.substring(0, 5);
// s2 is the catcher of the returned value
```

Object Basics

Class – What, When, Why, & How?

What:

- A blueprint for a custom **type**

When:

- Define a class when you're representing a concept (think nouns)
- When no other existing type can do what you want/need

Class – What, When, Why, & How?

Why:

- Keep similar concepts together
- Encapsulation (we'll expand on this next time)

How:

```
public class ClassName {
    // fields by convention go at the top
    // methods follow the fields
}
```

Using Objects and Methods

"Who does what, with what?"

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

- You've been using this in Java:

```
HashMap<String, Integer> scores = new HashMap<>();
scores.put("Bob", 78);

String name = "Bob Forapples";
int nameLen = name.length();
```

Constructors – What, When, Why, How?

What:

- Special method called when a new instance of a class is created
- Initializes the new instance
- Like the __init__ method in Python

When:

- Define a constructor when special initialization of a class is required
- Otherwise, Java implicitly creates a no-argument constructor if you don’t add one

Constructors – What, When, Why, How?

Why:

- Allows you to ensure that a new instance (object) of a class is a setup exactly how it needs to be before use of other methods/fields
- Puts the new object's internal fields in a good state by assigning meaningful values to each of the object's fields

How: **always has the same name as the class**.

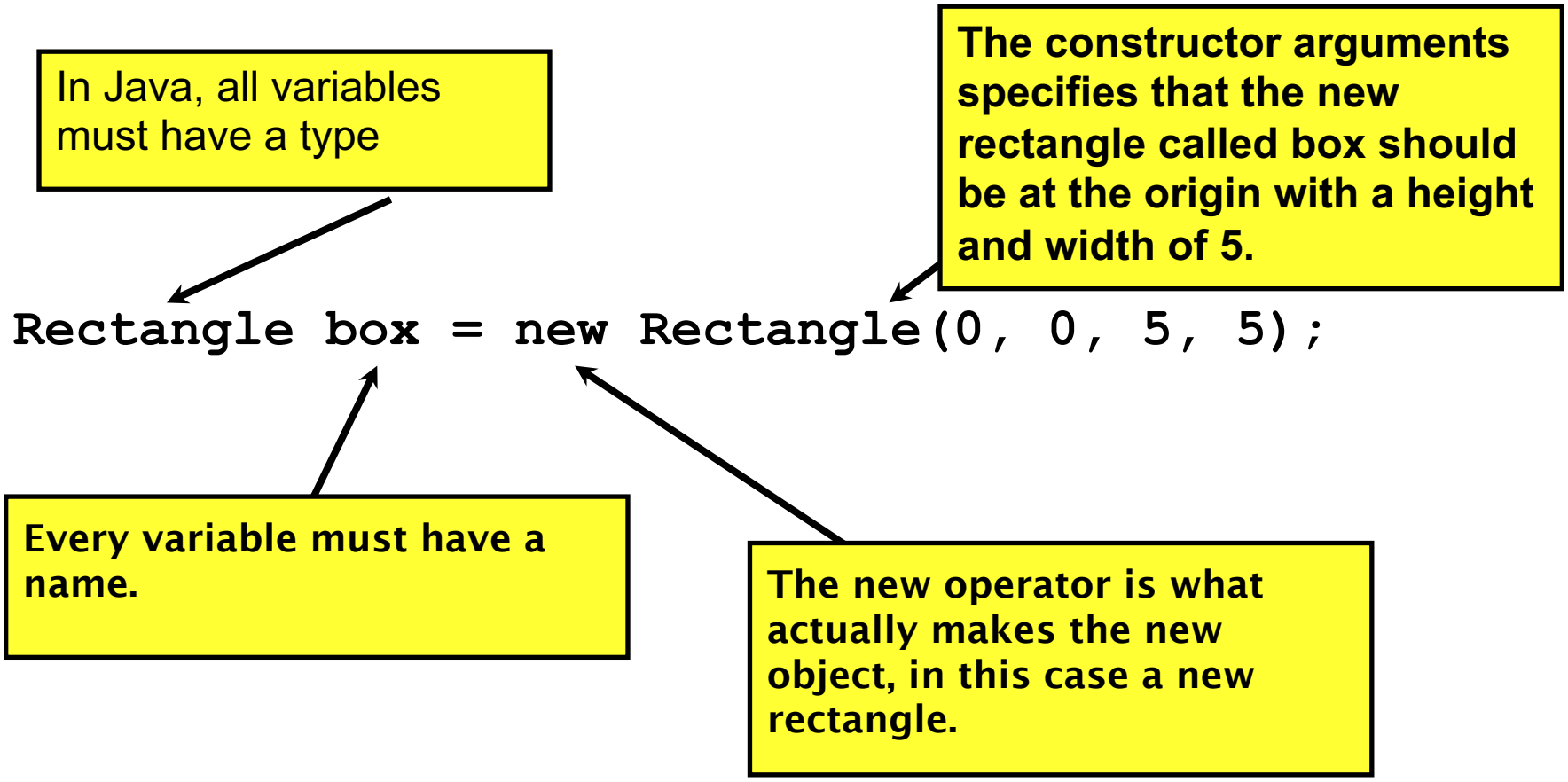
```
public class MyClass {
    public MyClass() {
        // no-argument constructor
        // initialization code
    }
    public MyClass(ParamType paramName) {
        // parameterized constructor
        // initialization code – typically uses paramName
        //    to initialize one or more of the internal fields
    }
}
```

Object Constructors

- int num = 5;
  - This works for primitive typed data

- What about “objects” (made from classes)?

Using Constructors



Q5

Object Constructors

- Open BankAccount.java
  - Let's do the first few, then work on your own
  - When you're done and it works, solve the last quiz question

Now code the StudentAssignments class yourself

- Uncomment the stuff in StudentAssignmentsMain to see what the class ought to do
- Then create the class and add the constructors and methods you need
- If you finish early, add a function to compute the student's average grade