

Object Intro and Miscellaneous

Import *ObjectIntroAndMiscPractice* project from git repo

Writing clean code

Comments are only the last resort

Functions

- Give functions descriptive names
- Don't make functions too long
- Rather than commenting an unclear function, modify the code so it is clear

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 conventions:
 - **variableNamesLikeThis**
 - **methodNamesLikeThis (...)**
 - **ClassNamesLikeThis**
- You should follow the conventions!

Javadoc comments

```
/**
 * 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) {
        ...
    }

    ...
}
```

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

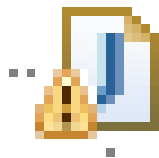
- Internal documentation for when someone reads the code itself
- External documentation for when someone re-uses the code

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:

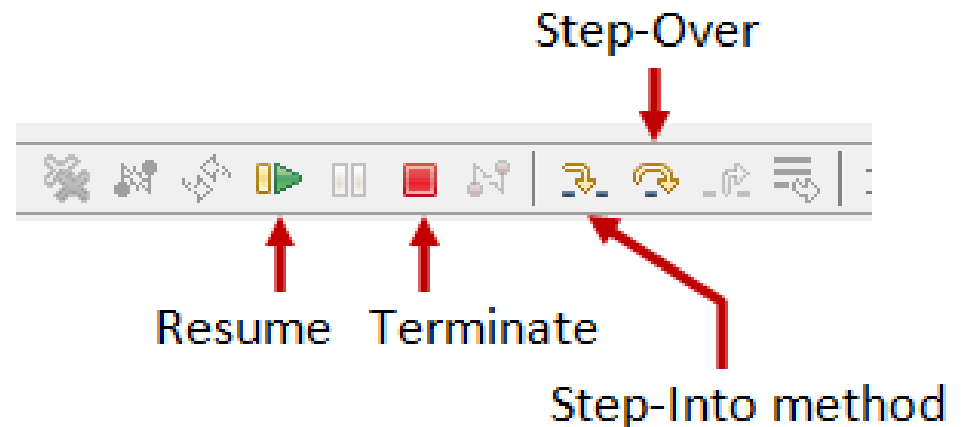
- See:
- https://www.rose-hulman.edu/class/csse/csse220/201920/Docs/grading_guide.html
- Write appropriate comments:
 - Javadoc comments primarily for classes.
 - Explanations of anything else that is not obvious in any spot.
- Give self-documenting variable and method names:
 - Use name completion in Eclipse, Ctrl-Space, to keep typing cost low and readability high
- Use Ctrl-Shift-F in Eclipse to format your code.



Debugging

Debugging—Demo

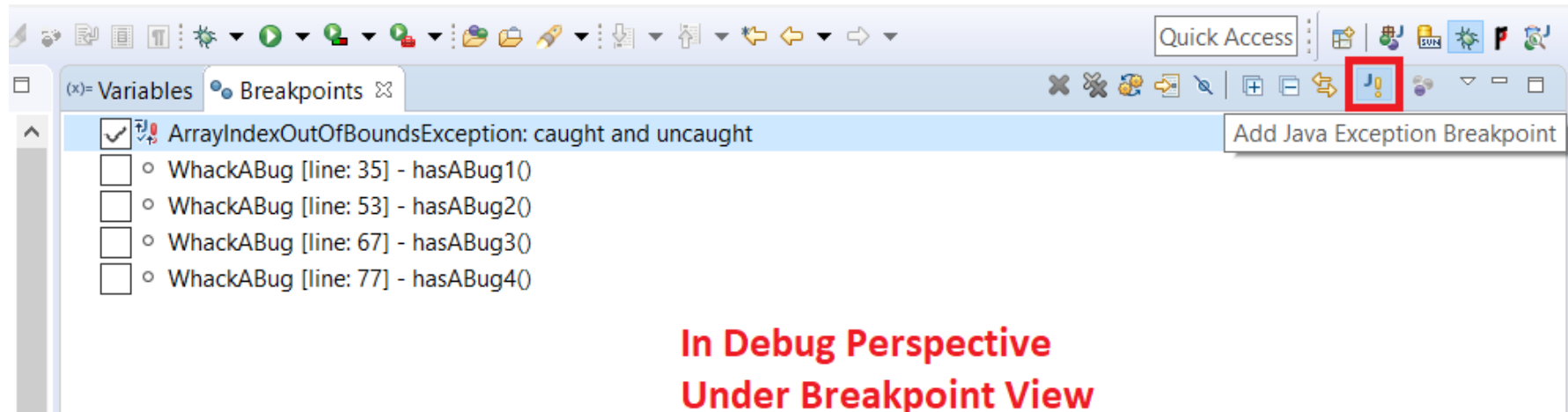
- ▶ Debugging Java programs in Eclipse:
 - Set a breakpoint where you want to start
 - Launch using the bug icon
 - Single stepping: *step over* and *step into*
 - Inspecting variables



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



**In Debug Perspective
Under Breakpoint View**

Important gotcha: Strings in java are immutable

- No method on the string class will modify the content of a string
- All methods instead return a new string

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  
    //methods  
}
```

Using Objects and Methods

“Who does what, with what?”

► Works just like Python:

- *object.method(argument, ...)*

Implicit
argument

Explicit
arguments

► 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 of a class is a setup exactly how it needs to be before use of other methods/fields
- Puts it in a good state

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

```
public class MyClass {
    public MyClass() {
        //initialization code
    }
    public MyClass(ParamType paramName) {
        //initialization code
    }
}
```

Object Constructors

- `int num = 5;`
 - This works for primitive typed data
- What about “objects” (made from classes)?

Using Constructors

In Java, all variables must have a type

The constructor arguments specifies that the new rectangle called box should be at the origin with a height and width of 5.

```
Rectangle box = new Rectangle(0, 0, 5, 5);
```

Every variable must have a name.

The new operator is what actually makes the new object, in this case a new rectangle.

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