# CSC 335 - Object-Oriented Programming and Design

Fall 2018
Dr. Jonathan Misurda
jmisurda@cs.arizona.edu

## Acknowledgements


- Lectures by Richard Mercer and Richard Snodgrass.
- Course design also by Andrew Oplinger.
- Many improvements by Ivan Vazquez


## This course

- CSC 335 is the fourth course in the introductory sequence of programming.
- This course explores:
  - Object-oriented design through the *Unified Modeling Language* (UML)
  - *Advanced Java object-oriented programming features* such as polymorphism through interfaces and inheritance, as well as event-driven programming and graphical user interfaces (GUIs) through JavaFX
  - *Design patterns*

## Concepts, Cont.

- You will, first individually and then in a pair, build a significant program in the first part of the course.
- In the final weeks, as part of a team you will design, implement, and test a complex system.
- You are expected to have previous Java programming experience and a knowledge of objects, classes, control structures (if...else, loops), arrays, and data structures.

## Learning Objective Levels

- Familiarity
- Guided usage
- Competence
- "Mastery"

## Eight Learning Objectives

1. Write programs in Java 8
   - (mastery of most of the language)
2. Use common Java APIs correctly and appropriately
   - (mastery of fundamental classes, competence for some others, and guided usage for advanced topics)
3. Compose UML class and sequence diagrams (mastery) and package diagrams (familiarity) to model the structure of a large Java program containing several dozen classes

## Eight Learning Objectives, cont.

4. Design, implement, and test a large (several thousand line) program exhibiting sophisticated logic, given a functional specification as user stories (competence)
5. Work productively and efficiently in a several-member team to realize the large program (guided usage)
6. Use modern programming tools competently in implementing a large Java program (mastery of fundamental skills to familiarity of advanced topics)

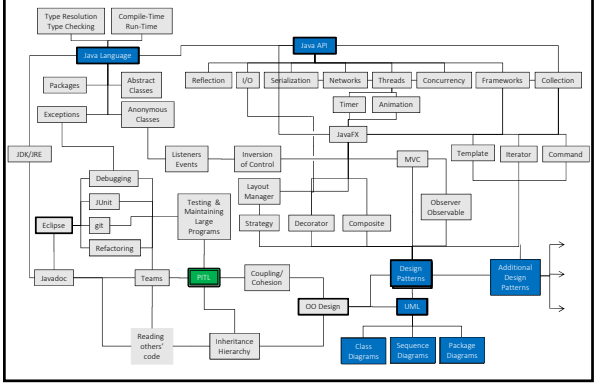
## Eight Learning Objectives, cont.

7. Understand common design patterns (guided usage for all) and use some of them in programs the students design and implement, including in the final project (competence in at least six)
8. Have some fun!

## Pedagogic Materials

- The Web
  - http://www.u.arizona.edu/~jmisurda/
  - Includes assignments, course notes, example code, other links
- Additional Handouts

## 335 Concept Map




## Course Moral

- We manage complexity through **abstraction**
- Abstraction allows us to hide the details of the implementation and usage from each other via an **interface**
- Complexity is an emergent property
  - Arises from the **composition** of simple parts, rather than intentionally creating something complex

## Java Programming Layers

| Java Source Code |
| --- |
| Java Compiler |
| Java Virtual Machine |
| Operating System |
| Hardware |

## Fractally Applied

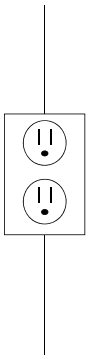
**Java Compiler**

| Lexical Analysis |
|---|
| Syntax Analysis |
| Semantic Analysis |
| Optimization |
| Code Generation |

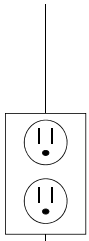
**Operating System**

| User Software |
|---|
| Device Independent Layer |
| Device Drivers |
| HW Interrupt Handlers |
| Hardware |

## Interface



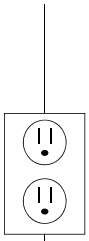
## Interface



Promise:

120 VAC
60Hz

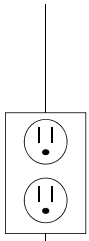
## Interface

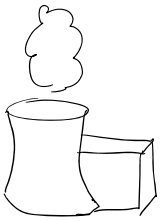


Promise:

120 VAC
60Hz

## Interface

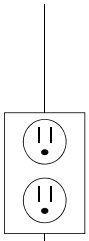


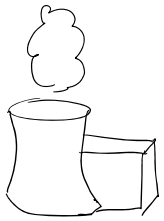


Promise:

120 VAC
60Hz

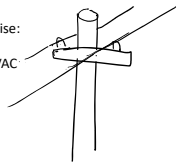
## Interface



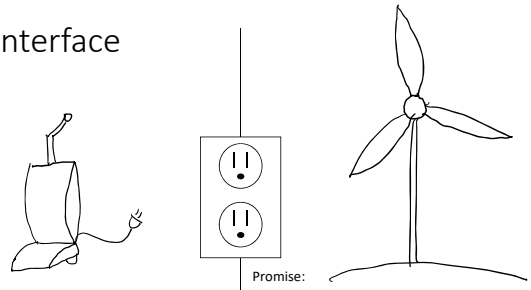


Promise:

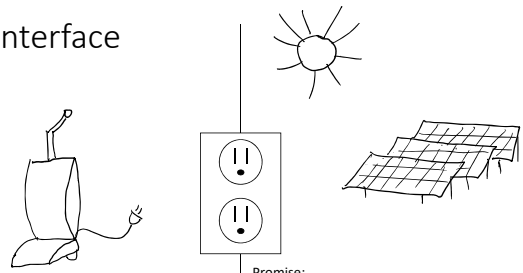
120 VAC
60Hz



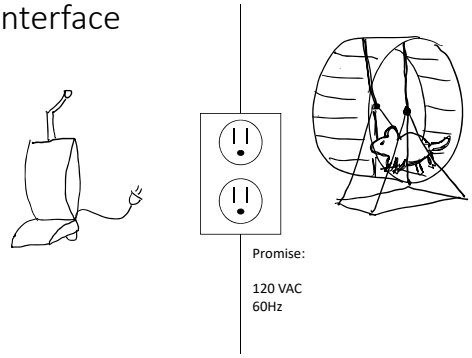
## Interface



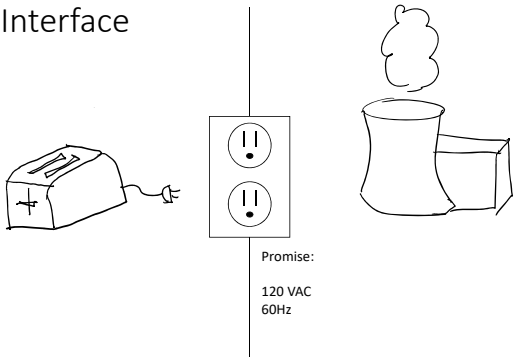

## Interface




## Interface




## Interface




## Group Exercise

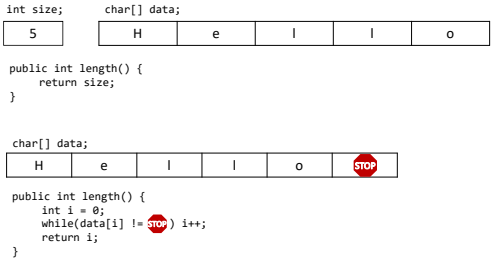
- Come up with a non-computer interface that you encounter in everyday life.
- What does it promise?
- What does it hide?

## Abstraction in Java

```
class String {
      …
      public int length();
      …
}
```

## Two possible implementations

```
String s = new String("Hello")
```




## Which implementation is it?

- We hope not to know
- If we *need* to know something about the implementation, our abstraction is insufficient and "**leaky**"
- All abstractions leak implementation at some level
  - How could we devise an experiment to determine which of the two String implementations Java uses?