Class #01–02:
Programs, Objects, and Methods

Software Design I (CS 120): M. Allen, 05–07 Sept. 17

Hardware: Central Processing Unit (CPU)

- The main electronic chip performing the computation
- The CPU can only usually perform very simple tasks
  - add/subtract/multiply two numbers at a time
  - compare two numbers to see which is larger
  - move/copy data from one spot to another
  - etc...
- The CPU appears much more powerful
  - This is because they do all of these things very quickly

Software: Computer Programs

- Software is a collection of data and instructions
- A program tells the CPU what to do
- A program must therefore “speak the language” of the CPU, telling it what to do in terms of a combination of the simple operations it can actually perform

A Basic Computing System

Software Design I (CS 120)
A programming language lets us:
1. Write down commands in a way that we can understand at a relatively sophisticated human level
2. Do so in a way that can be translated into “CPU-speak,” at the level of basic machine operations

A very great difference exists between those levels:
- Many degrees of possible language complexity
- Each language looks different and works somewhat differently at the high level of the programmer
- However, each gets “translated” into the same simple language of the CPU in order to run the program

High Level
- Close to Human Language

Low Level
- Close to CPU (add, subtract, etc.)

The Process in Java
- Things are slightly different here!
- We have another program, the Java Virtual Machine (JVM)
- The JVM can translate Java bytecode into CPU instructions
- In Java, the compiler step actually translates our instructions into bytecode, which is a combination of human-readable and formal constructions
  - Advantage: the same bytecode can run on any OS for which a JVM exists (which does its own final translation)
  - Disadvantage: someone has to write the JVM for each OS (the same thing is true for compilers, however)
  - Another (partial/possible) advantage: the JVM also seeks to verify the bytecode to avoid dangerous bugs and security holes (due to how complex programs may be, this is only partially possible)
The Java Programming Model

**Action**
- Programmer types in commands

**Tool**
- Text Editor
- Java Compiler
- Java Virtual Machine

**Result**
- Source code file (className.java)
- Bytecode file (className.class)

The program executes.

An Object
- This object is everyday, and yet complicated
  - **Interacts** with other objects
    - Cell towers
    - You!
  - **Made** of other objects
  - Most of us can figure out how to use it as a black box
    - That is, we can use it properly without knowing exactly how it does what it does

Object Classes
- Every Java object belongs to a class (group)
  - Tells us what kind of object it actually is
  - The membership in the group tells us something about the objects behavior and attributes
    - If you are given an existing class (written by someone else), and you want to use it in your own code, you need to know how to use it properly
    - Once you know how the class works, you can use any individual object from that class

A Software Object
- Like “real” objects, these can also be “black boxes”
- A piece of code, but also:
  - **Interacts** with other objects
  - **Made** of other objects
  - Has a state (what it’s like now)
  - Has behavior (things done to it or things done by it)
- Often, we can figure out how to use it as a black box
  - We can use it properly and innovatively without knowing exactly how it works
Class Diagrams

- Describes a class and how we can use it properly when writing our code
- Sketches all the properties and behaviors of objects that belong to the class
- No details about how this all works, just a list of what it does

<table>
<thead>
<tr>
<th>Class Name</th>
<th>attributes (instance variables)</th>
<th>operations (methods)</th>
</tr>
</thead>
</table>

- **DrawingGizmo**
  - **<<constructor>>**
    - DrawingGizmo()
  - **<<update>>**
    - void moveForward()
    - void turnClockwise()
    - void turnCounterClockwise()
    - void dontDraw()
    - void draw()
    - void delay2Sec()

Class Diagrams, cont’d.

- **DrawingGizmo** has no attributes given here, only methods for use
- That’s fine: it’s just a simpler object than some

- **No attributes!**

Class Diagrams, cont’d.

- **Constructor** method
  - Used to create an object
  - Has same name as class
  - Has no “return type” in front (unlike others that have **void**)

- **Constructor**
  - **<<constructor>>**
    - DrawingGizmo()
  - **<<update>>**
    - void moveForward()
    - void turnClockwise()
    - void turnCounterClockwise()
    - void dontDraw()
    - void draw()
    - void delay2Sec()

Class Diagrams, cont’d.

- **Update** methods
  - Things object can do
  - Usually change something about the object
  - Also called **mutators**
  - These return **void**, i.e., they have no output after they run (we’ll worry about this later on in the course)

- **Other methods**

- **Other methods**
  - **<<constructor>>**
    - DrawingGizmo()
  - **<<update>>**
    - void moveForward()
    - void turnClockwise()
    - void turnCounterClockwise()
    - void dontDraw()
    - void draw()
    - void delay2Sec()
Creating an Object in a Program

- To write a proper program, we must pay close attention to **syntax**
  - The rules of grammar, punctuation, spelling for Java
- Before we start using objects, we have to give them **names**, and our program must also tell the computer what **kinds of things** they are
  - Names must be **distinct** so the computer can tell which object is which
  - Spelling counts! An object named *“pen”* is **not** named *“Pen”*
  - **Meaningful** names are a good idea!
- Java syntax for naming a single object (**variable declaration**):
  
  ```
  className objectName;
  ```

  - className: the kind of object it is
  - objectName: the name we want to give the object

  - As an example, we **declare** a DrawingGizmo variable, “pencil”:
    
    ```
    DrawingGizmo pencil;
    ```

Creating an Object in a Program, cont’d

- For complex Java objects, picking a variable name is not enough
- To **use** a DrawingGizmo in a program, we must make sure we **actually create** such an object (**instantiation**)

  ```
  objectName = new constructorName();
  ```

  - objectName: the **variable name** we want to use for the object
  - new: a special Java word that tells computer to store new object in memory
  - constructorName: name of **constructor method** for the object (this is what builds the object: in Java, always the same as the class name itself)

  - Here, we **instantiate** DrawingGizmo object named “pencil”:
    
    ```
    pencil = new DrawingGizmo();
    ```

For This Week

- **Meetings this week:**
  - Tuesday/Wednesday and Thursday: regular classroom
  - Friday: in the CS Lab (16 Wing)
    - Important to be there to get oriented with the lab
    - You can bring your own computer if you like, but we will ensure everyone can log in to lab machines properly
    - Ahead of time: read Eclipse tutorials (class site, Resources Section)

- Please obtain the online text ASAP
- First reading due start of day, 7:45 AM Monday 11 September

- **Office Hours:** Wing 210
  - Thursday/Friday, 9:00 AM–10:30 AM