Introduction to Computers and Programming Languages

CS 180
Department of Computer Science
Purdue University
Reminders and Announcements

Course webpage:
- Syllabus
- Lecture and recitation slides are available
- Projects
- Course policies (note the integrity policy)
Class newsgroup:

- `purdue.class.cs180`
- Project information and answers to questions will be posted to the newsgroup; posts by grad TAs are official
- **You are responsible** for knowing all information posted, including changes to assignments

A newsgroup client such as Mozilla Thunderbird or Microsoft Outlook can access the class newsgroup.
Reminders and Announcements

- Discussion Groups/Mentoring
  - Facilitated by Debbie Perouli
  - Discussion of CS180 related material
  - When: Tue-Thur 5 to 7 in LWSN B116
  - Extra credit: 1/3 point (max of 5 total for the semester) for each session attended
  - More information to come later
Reminders and Announcements

Recitations
- A chance to ask questions
- Examples
- Review of/for exams
- Introduction/solutions to assignments

Plenty of help is available for this course. Make sure to take advantage of it if you need to.
Reminders and Announcements

Projects

- Generally 1 per week or equivalent.
- Released on course web page
- Each led by 1 TA
- Consulting available in the lab, mon, tue and wed 7-10 pm.
- Make **sure** your project compiles. If it doesn't, you will get a 0.

Book

- Read it, if something is unclear, ask.
- You will be surprised how useful it can be.
Recall the Basics

- Hardware - physical devices (CPU, disks)
- Software - information *stored* in hardware
  - Instructions - executed by the CPU
  - Data - manipulated by CPU

- A **program** is a list of instructions for the CPU used to achieve a goal. Similar to:
  - recipes used to create some food
  - designs used to construct a model
  - driving instructions to reach a destination

- Input / Output
Recall the Basics

- Computer science is about analyzing problems and information.
  - In here, we do this through creating programs
  - We must properly express and model the problems that we analyze.
Recall the Basics

- To create programs, we use a **programming language**
  - Human readable format for computer instructions
  - Written in text files (source code) and converted (compiled) into a usable form (e.g. machine language)

- In this course we will work with the **Java programming language**
  - purely Object-Oriented. (Modeling!)
  - compiled to Byte Code.
  - interpreted by the JVM
Data

- All data in a high level language has a type.
  - What *kind* of thing does the data represent?
  - Numbers, Text, Cats, Dogs, Weevils

- Every piece of data is stored at some place in memory.

- An **object** is a collection of some data along with pieces of code that can manipulate that data. (Simplify program design)
Classes and Objects

- An object represents a discrete concept, such as a particular Student, Car, or ATM.

- Similar objects share characteristics and behavior. They are grouped into a class.
  - (Professor Y and Professor Z are both professors)

- A class defines the data and behaviors of an object.
  - A class is a cookie cutter; an object is a cookie
  - The Platonic ideals and their instantiations!
    *(Applied philosophy)*

- An object is called an instance of a class.
Graphical Representation of a Class

The notation we used here is based on the industry standard notation called **UML**, which stands for Unified Modeling Language.

We use a rectangle to represent a class with its name appearing inside the rectangle.

**Example:**

- Account
- Motorcycle

The notation we used here is based on the industry standard notation called **UML**, which
Graphical Representation of an Object

We use a rectangle to represent an object and place the underlined name of the object inside the rectangle.

Example:

This is an object named SV198.
An Object with the Class Name

This notation indicates the class which the object is an instance of.

Example:

SV198 : BankAccount

This indicates that object SV198 is an instance of the BankAccount class.
Messages and Methods

- To instruct a class or an object to perform a task (behavior), we send a \textit{message} to it.
  - (Tell a car to drive forward)

- A class or an object must possess a \textit{matching method} to be able to handle the received message.
  - (A car has no method to hover, so it cannot respond to a hover message)

- A value we pass to an object when sending a message is called an \textit{argument} of the message.
  - (Tell a car to drive forward 3 miles)
What is encapsulation, and how do methods and objects help provide it?
Sending a Message

Message deposit with the argument 250.00 is sent to a BankAccount object SV198.

deposit (250.00) ➔ SV198 : BankAccount
Sending a Message and Getting an Answer

Ask for the current balance of this particular account.

getCurrentBalance()

SV198 : BankAccount

current balance

The current balance of SV198 is returned.
Calling a Class Method

Ask for the maximum possible speed for all MobileRobot objects.

MobileRobot

getMaximumSpeed()

maximum speed

How do class and object methods differ?
An object comprises data values and methods.

An instance data value is used to maintain information specific to individual instances
- each BankAccount object maintains its own balance

A class data value is used to maintain information shared by all instances or aggregate information about the instances
- E.g. minimum balance
- Reduces duplication, prevents errors
Sample Instance Data Value

All three BankAccount objects possess the same instance data value current balance.

The actual dollar amounts are, of course, different.
Sample Class Data Value

There is one copy of minimum balance for the whole class and shared by all instances.

This line is an instance-of relationship.

SV129 : BankAccount

current balance
908.55

SV098 : BankAccount

current balance
1304.98

SV211 : BankAccount

current balance
354.00
When the class icon is not shown, we include the class data value in the object icon itself.

SV129 : BankAccount

- **minimum balance**: 100.00
- **current balance**: 908.55
Inheritance

- **Inheritance** is a mechanism in OOP to design two or more entities that are different but share many common features.
  - Features common to all classes are defined in the *superclass*.
  - The classes that inherit common features from the superclass are called *subclasses*.
- We also call the superclass an *ancestor* and the subclass a *descendant*. 
A Sample Inheritance

Here are the superclass **Account** and its subclasses **Savings** and **Checking**.
Inheritance (cont.)

- The account class may define
  - Owner’s Name(s)
  - Getbalance

- The Checking class may define
  - Minimum (checking) balance
  - ATM transactions

- The Savings class may define
  - Minimum (savings) balance
  - Interest rate
  - Pay interest
Inheritance Hierarchy

- An example of an inheritance hierarchy among different types of students.

Diagram:

- Student
  - Graduate
    - Masters
    - Doctoral
    - Law
  - Undergrad
    - Commuting
    - Resident
Software Engineering

- Much like building a skyscraper, we need a disciplined approach in developing complex software applications.
- **Software engineering** is the application of a systematic and disciplined approach to the development, testing, and maintenance of a program.
- In this class, we will learn how to apply sound software engineering principles when we develop sample programs.
Software Life Cycle

- The sequence of stages from conception to operation of a program is called the \textit{software life cycle}.

- Five stages are
  - Analysis
  - Design
  - Coding
  - Testing
  - Operation and Maintenance

Remember these during projects!
Quiz

1. In 6 lines or less, describe to a computer how to put on a belt.

2. If my program does not work as I expect it to, what should I do?
   a. Give up, its not worth the time.
   b. Blame my TA.
   c. Throw my computer out of the window and get my parents to buy me a new one.
   d. Take a deep breath, clear my head, and try to figure out what I did wrong.