PEER-LED TEAM LEARNING
COMPUTER SCIENCE

MEETING 4 – STUDENT VERSION
DEFINING CLASSES

BARBARA G. RYDER AND PRADIP HARI

Exercise 1: Simulating objects in real life
In this exercise, we will practice defining classes for real life situations that we could simulate using object-oriented programming. This means we have to figure out what to use as objects and how to define their properties and the natural operations on them.

Key questions in formulating an object-oriented program:

What are the objects you might need?
What operations would be performed on those objects?
Do these objects have properties or attributes with values?
What sequences of operations on these objects "make sense" (i.e., how to test ourclass definition?)

a. Think about simulating a car (and its systems) by an object-oriented program.

- Have one student portray the Car object itself.
- Another student should suggest an operation that could be performed on (or by) a car and act it out. Try to think of at least 5 operations for your Car object. Write each operation on the board.
- Now think of properties or attributes that a Car object might have. Using post-it notes, label the car with its properties (e.g., color).
- Finally, think of sequences of operations that 'test' that your Car runs properly; write them on the board. Did some of your operations require you to define more properties for your Car object?

b. Now think about writing an Appointment class. What operations are possible on Appointment objects? What sorts of operations would be needed for Appointments? What inputs and outputs are necessary for these operations? What data is involved in an Appointment?

- Take turns answering these questions in the group, and writing your answers on the board. You should try to get at least 3 operations and 3 fields for this class.

- Taking turns, write your operations so that they are legal Java method signatures. *A method signature* is the first statement in the code for a method, which states its return type, its name, and gives the types and names of its parameters in order.

- Taking turns, define Java fields for the data an Appointment object needs.

- Are there other behaviors of Appointments that are not covered by your methods? Name one. How would you provide that behavior?

Exercise 2: Defining a class and flow charting its methods

Now think of simulating a bank account by a Java program. Again, take turns contributing answers to these questions and write them on the board.

- What objects would you need to define to simulate a bank account? (Think about what people do with bank accounts.)
- What properties or attributes would those objects have?
- What operations "make sense" on the objects in your simulation?
- Think of 2 sequences of these operations that would be good to use to 'test' the functionality of the methods in your class. Justify your choice.
- Can you think of a sequence of operations for bank accounts that might cause problems?

Now let's think of more complicated bank accounts – such as those which might require a minimum balance to avoid per check fees.

- Add two things to your previous bankAccount class to simulate a more complicated account
- What new methods will you need?
- What previous methods will need to be changed? How?
- Choose two of the changed methods and for each, draw its flow chart (as a group) on the board. Take turns adding steps to the flow chart.
- Now think about adding error checking to your flow chart.
- Now think about how to test the code that will be written for your algorithm. What inputs do you need and what outputs do you expect?

Exercise 3. Battleship game.
Choose a partner and play the battleship game your peer leader has brought to class. You will play for a short time to understand what will be necessary to simulate this game in a computer program. There are 2 boards one for each player (which cannot be seen by her opponent). The idea of the game is to guess where your opponent's ships are and sink them. To win one must sink all of her opponents ships.

Basic Rules: The board is divided into columns labeled by letters and rows labeled by numbers. Each player receives *k* ships and places them on his board. The players take turns guessing the position of their opponent's ships. The opponent must answer "hit" or "miss" depending on whether or not the player guessed a square occupied by the opponent's ship or not. Each guess is recorded on the player's own board so she can remember if it was a 'miss' or 'hit' (and avoid redundant guesses.) Ships vary in size, but are rectangular.

Each pair of students should prepare answers to the following questions after playing the game.

What objects are necessary to simulate the Battleship game?
What are their properties or attributes?
What operations are necessary for the game?
How do we know when someone wins?

Exercise 4. Interacting classes

Imagine that you are watching your TV using your remote, and you decide to watch a DVD instead. Let one student be a TV object, one a DVD-player object and on be a Remote object. Now have these three students step through how you would watch a DVD, starting with all the equipment turned off. After exploring this simulation, then replay your simulation while answering the following questions:

- What are the objects in this simulation?
- What are the possible interactions between these objects? Which operations in one class have to call operations in the other class? Why?
- Take turns writing the operations on the board. Note when one operation calls another, and when these are in the same class or different classes. Recall that when we define operations we think about inputs, outputs and error conditions.
- How would we test these operations?
- What would be different if your task was to simulate recording to a DVD or VCR?

Cite this module as: Ryder, B.G., Hari, P. (2012). Peer-Led Team Learning Computer Science: Meeting 4 - Student Version; Defining Classes. Online at http://www.pltlis.org.
Originally published in *Progressions: The Peer-Led Team Learning Project Newsletter*, Volume 9, Number 1, Fall 2007.