CS115 Week 5
Code Refactoring, Object Interaction, Loops

BRING A FLASH DRIVE WITH:
Your current TRICK THE TURTLE scenario
Book scenario, chapter 10, breakout

Greenfoot engine

- A class with no `act()` method means the objects of that class will not be controlled by the Greenfoot engine.

Code Refactoring

- Breaking down long methods so each method has only one function. Usually entails adding private methods.
- Recognizing repeated code that could be turned into a method with parameters
- Thinking about attributes and methods of functions to ensure they are places correctly, especially considering object interactions (cohesion)

Object Interaction

- Objects are able to communicate only if one object has the reference to the other object.
- Approaches in Greenfoot are:
  - Pass the reference of one object to the other when constructing the second object (or make a World method to get it)
  - Call Actor methods
    - `getIntersectingObjects(java.lang.Class cls)`
    - `getNeighbours(int dist, boolean diagonal, java.lang.Class cls)`
    - `getObjectsAtOffset(int dx, int dy, java.lang.Class cls)`
    - `getObjectsInRange(int radius, java.lang.Class cls)`
    - `getOneIntersectingObject(java.lang.Class cls)`
    - `getOneObjectAtOffset(int dx, int dy, java.lang.Class cls)`
  - All Actor objects can call `getWorld()` and call World methods
    - `getObjects(java.lang.Class cls)`
    - `getObjectsAt(int x, int y, java.lang.Class cls)`

Object Interaction (cont)

In `MyWorld` constructor, difference between
- local variables for instantiating objects (and then `addObject()` to World). The local variable goes away when the constructor/prepare ends, but the World still has it, but it is hard for us to get at it (World methods are limited unless objects are touching). This is called "coupled", World is intermediary.
- instance variables in `MyWorld` could also be used to enable objects in the world talk to each other with are construction time or later (uncoupled)

Two types of loops

- count-controlled loops
  repeat a specified number of times
- event-controlled loops
  something happens inside the loop body that causes the repetition to stop after the current iteration completes
Count-controlled Pattern
An initialization of the loop control variable
An expression to test for continuing the loop
An update of the loop control variable to be executed within each iteration of the body

```c
int loopCount;              // Declare loop variable
loopCount = 1; // Initialize loop variable
while (loopCount <= 10) // Test expression
{
    . // Repeated actions
    .
    loopCount = loopCount++;  // Update loop variable
}
```

Do Statement
Is a looping control structure in which the loop condition is tested after executing the body of the loop

```c
do {
    Statement(s);
}
while(Expression);
```
Loop body can be a single statement or a block

Do vs. While
- POSTTEST loop(exit-condition)
- The loop condition is tested after executing the loop body
- Loop body is always executed at least once
- PRETEST loop(entry-condition)
- The loop condition is tested before executing the loop body
- Loop body may not be executed at all

“for” statement - Designed for Count-Controlled Loops
SYNTAX

```c
for(initialization; test expression; update)
{
    0 or more statements to repeat
}
```

Event-controlled
- Sentinel controlled - Keep processing data until a special value which is not a possible data value is entered to indicate that processing should stop
- End-of-file controlled - Keep processing data as long as there is more data in the file
- Flag controlled - Keep processing data until the value of a flag changes in the loop body

Loop Testing and Debugging
- Test data should test all sections of the program
- Beware of infinite loops -- the program doesn’t stop
- Check loop termination condition, and watch for an OBOB (off-by-1 bug)
- Use algorithm walk-through to verify that appropriate conditions occur in the right places
- Trace execution of loop by hand with code walk-through
- Use a debugger (if available) to run program in “slow motion” or use debug output statements