Lecture 9:
Introduction to Objects and Classes

Last time:
1. Precedence and short-circuiting
2. Type casting
3. Software lifecycle

Today:
1. Pseudo-code (from last lecture)
2. Objects and classes
3. Heaps
4. Garbage Collection
Questions

● What is `System in System.out.println()`?
● Why use `str.equals("cat")` to compare equality of `String str` and "cat"?
● Is the similarity of the notations `System.out.println()` `str.equals()` `sc.nextInt()` important, or coincidental?
Answers

- … will come, but are based on fact that Java is an object-oriented (OO) language
  Other OO languages: C++, Smalltalk, Eiffel
- What is an “object”?
- What does it mean for a language to be object-oriented?
Objects

- Bundles of (related)
  - data ("state")
  - operations ("behavior")
- Data often referred to as instance variables
- Operations usually called methods
- Invoking operations can change state (values stored in instance variables)
Example of Objects

- Bank account
  - State: account number, type, owner, balance, etc.
  - Operations: deposit, withdrawal, transfer, balance inquiry, etc.

- Student
  - State: name, ID, date of birth, major...
  - Operations: lookup grades, lookup class, compute tuition, compute age, …

- Scanner
  - State: stream to read from (e.g System.in)
  - Operations: get an int, double, byte, …

- String
  - State: sequence of characters
  - Operations: comparison test, equality test, (much more), …
Sample Student Object

<table>
<thead>
<tr>
<th>State</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>getAge</td>
</tr>
<tr>
<td>ID</td>
<td>date → age</td>
</tr>
<tr>
<td>DOB</td>
<td>getGrades</td>
</tr>
<tr>
<td>Major</td>
<td>semester → grades</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>

- Name: Kerry Keenan
- ID: 444230695
- DOB: 06-22-1987
- Major: CMSC
Accessing State / Methods

- If
  - o is an object
  - v is an instance variable of the object
  - m is a method of the object
- Then
  - o.v is how to access the data in v
  - o.m is how to invoke m
- So
  - System is an object, with out an instance variable
  - out is also an object, with println a method
  - System.out.println is how to access this method!
- Suppose str is a String
  - str is an object!
  - Methods of this object: equals, compareTo, etc.
  - str.equals, str.compareTo, etc. invokes these objects
Object-Oriented Programming

- Programs are collections of interacting objects
- Writing programs involves identifying what the objects should be and programming them
- Object-oriented languages provide features to ease object-oriented programming
- Defining objects involves indentifying
  - state
  - methods
More Questions

- How do you create objects?
- Do objects have “types”?
- Answer: classes
Classes

- “Blueprints” (“templates”) for objects
- Classes include specifications of
  - Instance variables (including types, etc.) to include in objects
  - Implementations of methods to include in objects
- Classes can include other information also, as will be seen later
  - static methods / instance variables
  - public / private methods, instance variables
  - And so on
Student Class Example

Conceptually:

- **Instance variables:**
  - String name
  - int ID
  - int dateOfBirth
  - String major

- **Methods**
  - getAge
  - getGrades
  - etc.

- The actual class implementation will include code for the methods
- This describes a blueprint for student objects
- How do we create objects from such a class (blueprint)?
How Are Objects Created?

- In Java: using `new`
  Recall:
  ```java
  Scanner sc = new Scanner (System.in);
  ```
- Invoking `new`:
  - creates fresh copies of instance variables in the "heap"
  - returns the address where the fresh variables are stored
- Heap? Address?
Heap = “Fresh Memory”

- While a program is running, some memory is used to store variables
  - Terminology: stack
  - We have been representing stack as table, e.g.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>4.5</td>
</tr>
</tbody>
</table>

- Rest of memory is called heap and can be used for other purposes, including storing new objects
Main Memory

- Stack grows, shrinks during program execution (why?)
- So does “allocated heap” (part of heap in use)
- Unallocated part of heap is called “free”
Object Creation

- New space allocated in heap to store instance variables
- Reference (= address) to this space is returned

Scanner sc = new (...);
Strings Are Objects

- Oh really? Where is `new` in
  
  `String name = “Narita”;`?

- Java provides it!
  - `String` is special because it is used so often
  - Java automatically “fills in” `new`
  - You can too:
    
    `String name = new String(“Narita”);`
In Java, 9 Sorts of Variables

- 8 primitively typed
  - Types are the 8 built-ins (int, byte, double, etc.)
  - Storage allocated on stack based on type
  - Value stored in stack
  - e.g. int x

- Reference typed
  - Types are classes
  - Storage allocated on stack to hold one memory address (typically, one word)
  - What is stored in stack is reference to heap, where actual data is stored
  - e.g. Scanner sc = new Scanner (System.in);
Example

```java
int x = 7;
float y = 3.3;
String f = "cat";
```
Heap Issues

- What happens if `new` is called and there is no free heap? Crash!
- What happens if following are executed?
  ```java
  String s;
  s = new String("cat");
  s = new String("dog");
  s = new String("cow");
  ```
- Wasted heap
  - “cat”, “dog” no longer referenced by stack
  - Crashes become a problem!
Garbage Collection

- This “heap management” or “memory management” issue is central in CS
- Java copes by invoking garbage collector to reclaim unused but still-allocated heap space
- Garbage collector reclaims memory in allocated heap and returns it to free heap
- In previous example, “cat” and “dog” would be reclaimed