CSS 142

Lecture 10

aretik@uw.edu

TODAY'S CONTENT

1. File I/O: Path Names; More on Exceptions
2. Intro to OOP
3. Classes
4. Class Activities: Read HW 5

First half++

Read HW 5

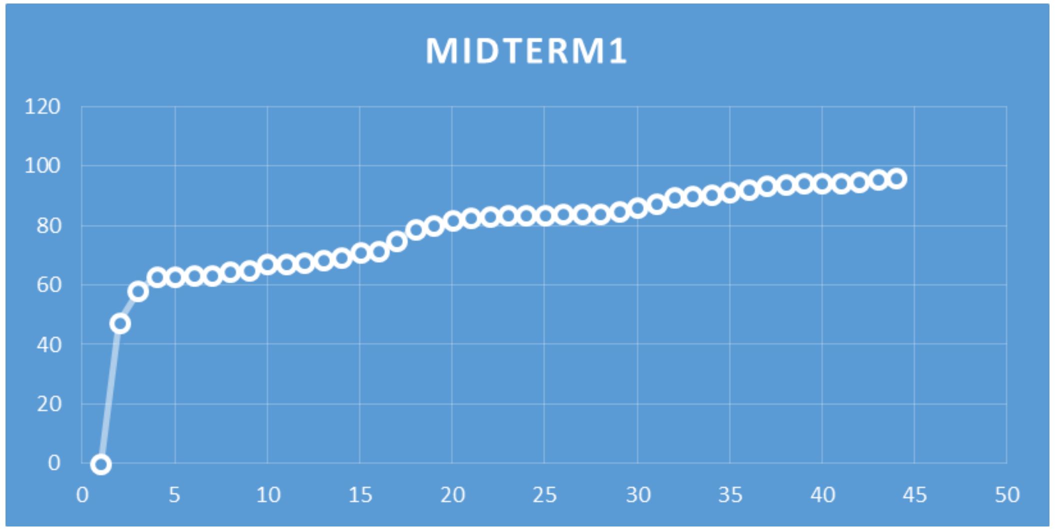
Wed NO CLASS: instead – ETHIC ASSIGNMENT 1 & 2 – self work | home

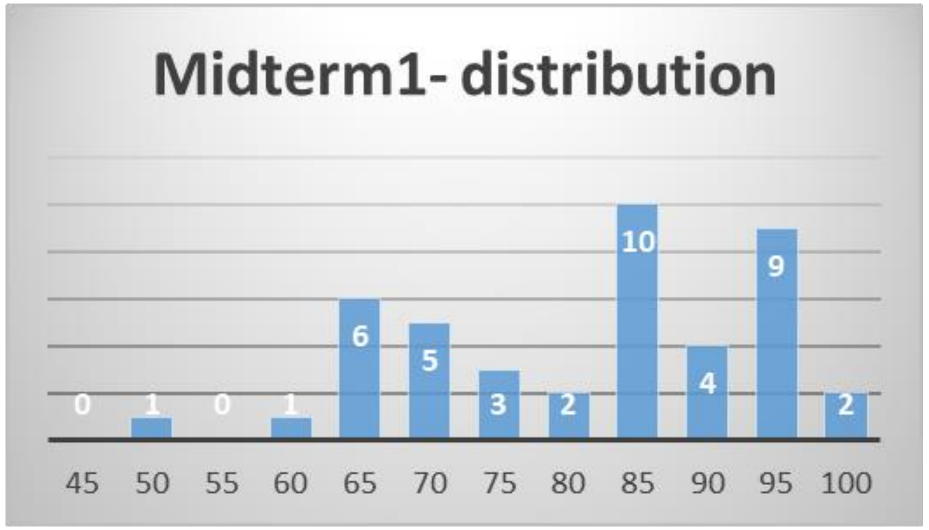
- see Lecture 11 notes

Next week reading :

- Mon
  - 4.3; 4.4 && Arrays 6.1; 6.2

Midterm 1


MIDTERM1
120
100
80
60
40
20
0
0
5
10
15
20
25
30
35
40
45
50


Midterm1- distribution
0
1
0
1
6
5
3
2
10
4
9
2
45
50
55
60
65
70
75
80
85
90
95
100

File I/O: Path

Path Names

- When a file name is used as an argument for opening a file, it is assumed that the file is in the same directory or folder as the one in which the program is run
- If it is not in the same directory, the **full** or **relative path name** must be given

Path Names

demo

- A *path name* not only gives the name of the file, but also the directory or folder in which the file exists
- A ***full path name*** gives a complete path name, starting from the **root** directory
- A ***relative path name*** gives the path to the file, starting with the directory in which the program is located

Drive name

U:\0-CSSSKL162AB-Aut2015\Lab4\Debugging\Solutions\Erik\hw2.java

"Lab4\Debugging\Solutions\Erik\hw2.java"

Write a relative path for a program running in lab4

Path Names

- The way path names are specified depends on the operating system
  - A typical **Unix path** name that could be used as a file name argument is

    `"/user/aretik/data/data.txt“`

  - A `BufferedReader` input stream connected to this file is created as follows:

```
BufferedReader inputStream =
  new BufferedReader(new FileReader("/user/aretik/data/data.txt"));
```

What the difference b/w Unix (aka UNIX) and Linux?

Path Names

- The Windows operating system specifies path names in a different way
  - A typical **Windows path** name is the following:

```
C:\dataFiles\goodData\data.txt
```

  - A **`BufferedReader`** input stream connected to this file is created as follows:

```
BufferedReader inputStream = new
  BufferedReader(new FileReader ("C:\\dataFiles\\goodData\\data.txt"));
```

Why \\ and not \

  - Note that in Windows **`\\`** must be used in place of **`\`**, since a single **backslash** denotes an the beginning of an escape sequence

Path Names

- A double backslash (\\\\) must be used for a Windows path name enclosed in a quoted string
  - **This problem does not occur with path names read in from the keyboard**
- Problems with escape characters can be avoided altogether by always using Unix conventions when writing a path name
  - **A Java program will accept a path name written in either Windows or Unix format regardless of the operating system on which it is run**

`System.in`, `System.out`, and `System.err`

- Using these methods, any of the three standard streams can be **redirected**
  - For example, instead of appearing on the screen, error messages could be redirected to a file
- In order to redirect a standard stream, a new stream object is created
  - Like other streams created in a program, a stream object used for **redirection must be closed** after I/O is finished
  - Note, **standard streams do not need to be closed**

`System.in`, `System.out`, and `System.err`

- Redirecting `System.err`:

```
public void getInput()
{
  . . .
  PrintStream errStream = null;
  try
  {
    errStream = new PrintStream(new FileOutputStream("errMessages.txt"));
    System.setErr(errStream);
    . . . //Set up input stream and read
  }
```

`System.in`, `System.out`, and `System.err`

```
    catch(FileNotFoundException e)
    {
      System.err.println("Input file not found");
    }
    finally
    {
      . . .
      errStream.close();
    }
  }
```

More on Exceptions:

terminology

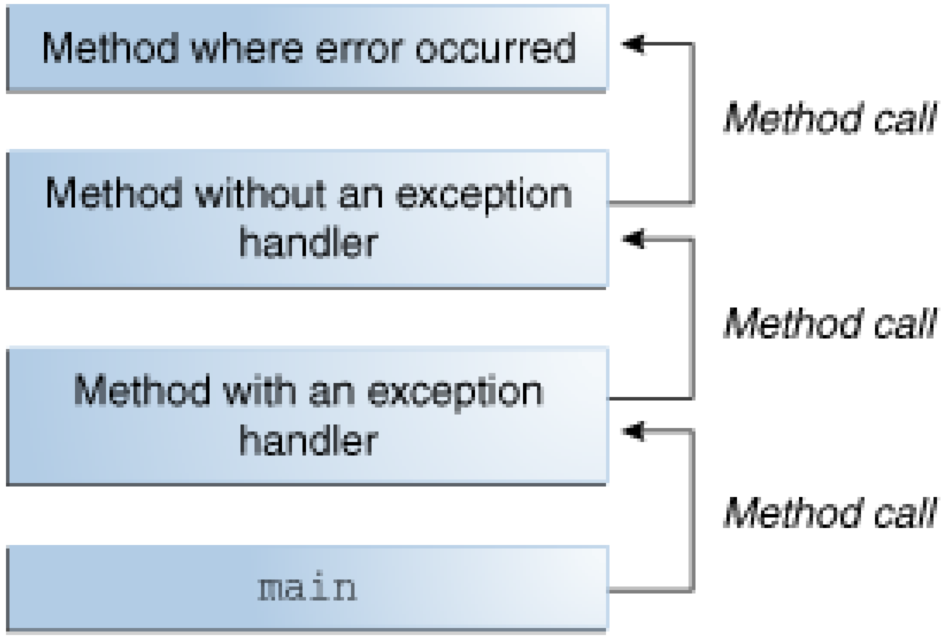
What Is an Exception?

The term *exception* is shorthand for the phrase "exceptional event."

> **Definition:** An ***exception*** is an **event**, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the **method creates an object and hands it off to the runtime system**. The object, called an ***exception object***, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called ***throwing an exception*.**

After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred. The list of methods is known as the ***call stack*.**


Method where error occurred
Method without an exception handler
Method with an exception handler
main
Method call
Method call
Method call

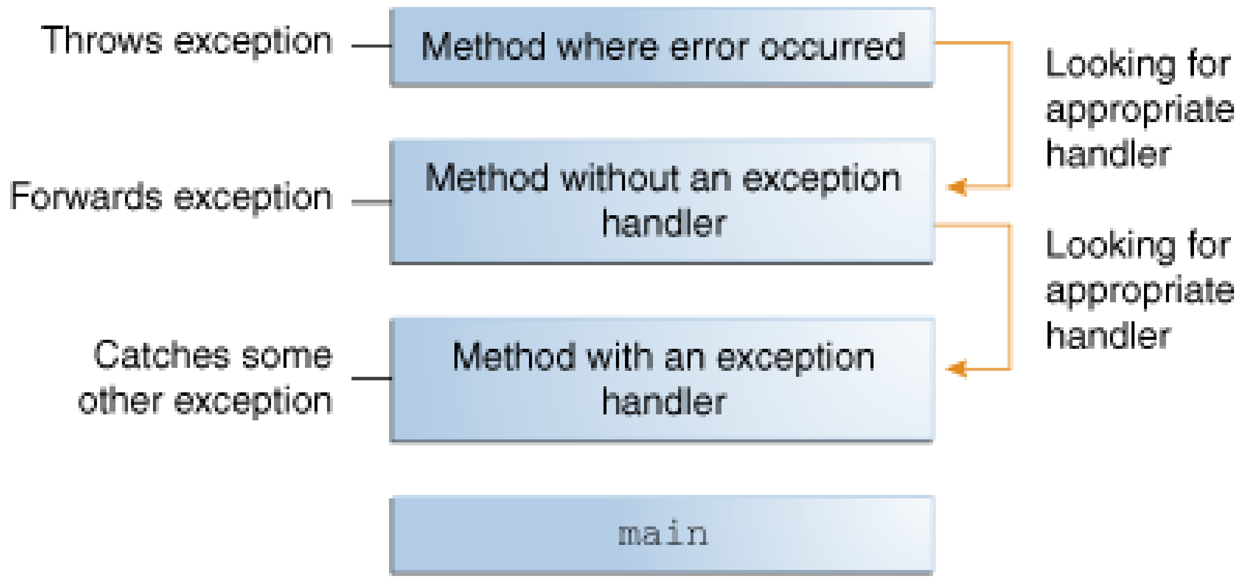
Call Stack

Exception Handling

The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code is called an ***exception handler***.

The search begins with the method in which the error occurred and proceeds through the call stack **in the reverse order** in which the methods were called. When an appropriate handler is found, the runtime system passes the exception to the handler. An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler.

The exception handler chosen is said to ***catch the exception***. If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, the runtime system (and, consequently, the program) terminates.


Throws exception
Method where error occurred
Looking for appropriate handler
Forwards exception
Method without an exception handler
Looking for appropriate handler
Catches some other exception
Method with an exception handler
main

Searching the call stack for the exception handler.

Intro to OOP

Java Classes

Defining Classes

Fundamental Building Blocks of Programs

STATE / DATA

- **variables**

  memory location/container

- **types**

  type of data

members

BEHAVIOUR / METHODS

- **control structures**

  loops and branches

- **subroutines**

  methods & functions

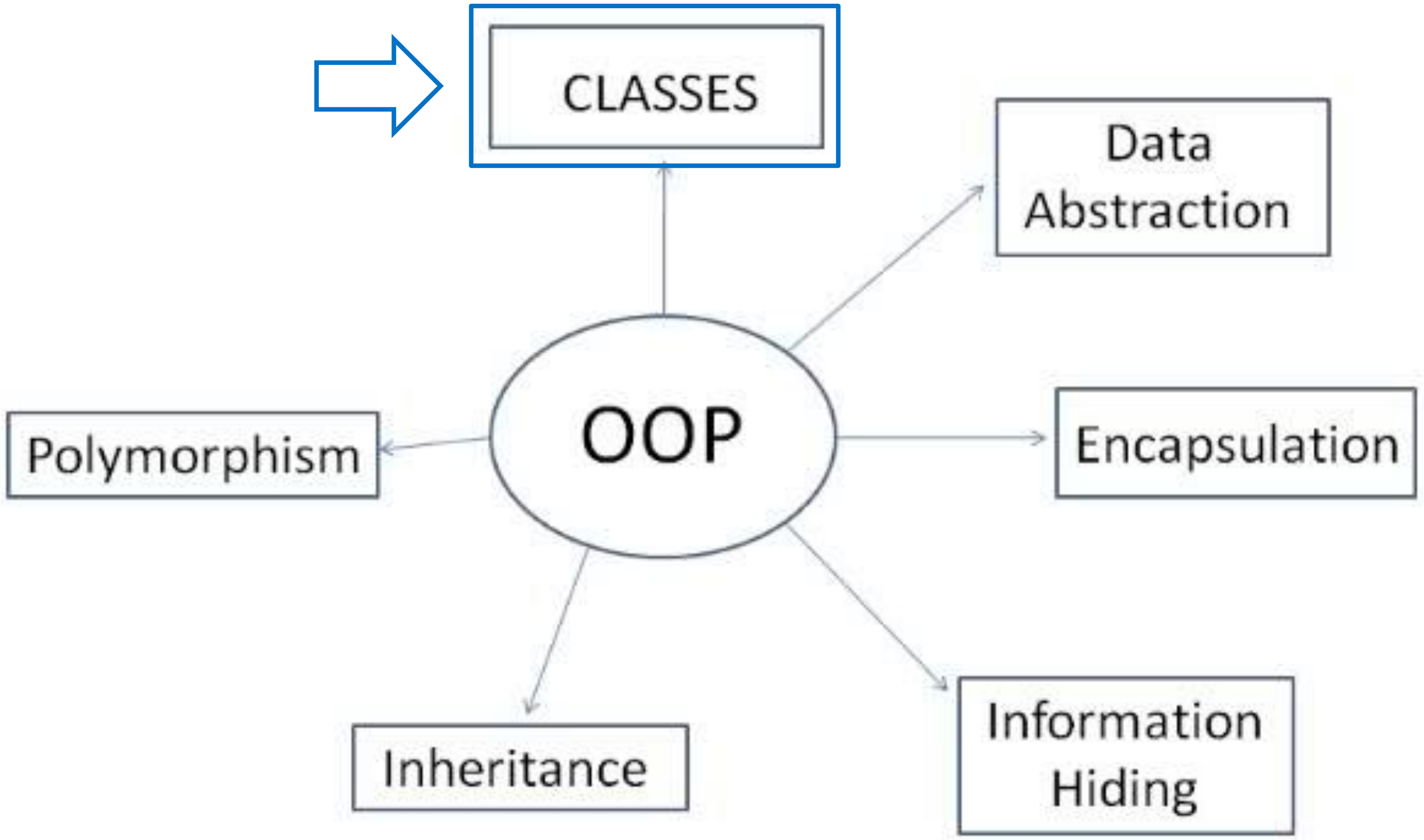
class

OOP *structure/tools to deal with* ***complexity*** *(using classes / objects)*

Syntax and Semantics

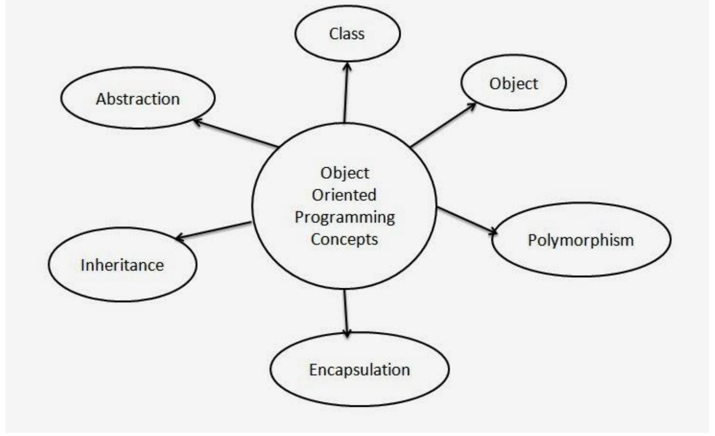
- *the syntax describes* ***how*** *you write a program and*
- *the semantics describes* ***what*** *happens when you run the program.*

Object –Oriented Programming (OOP)



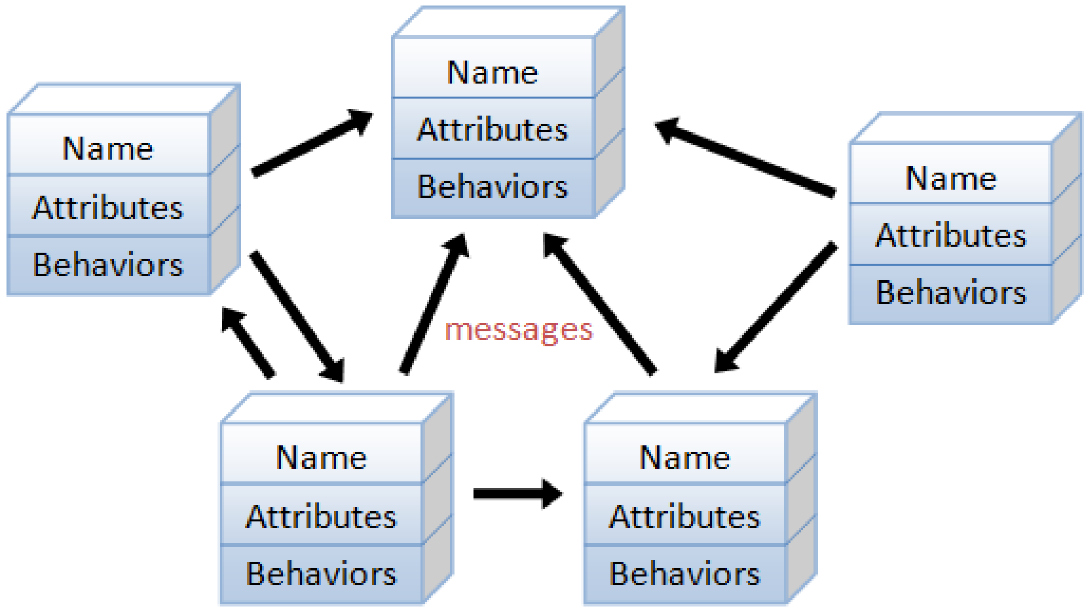
Where the OBJECTS?

OOP from 10,000 feet
(or from one Main () to many Mains ())

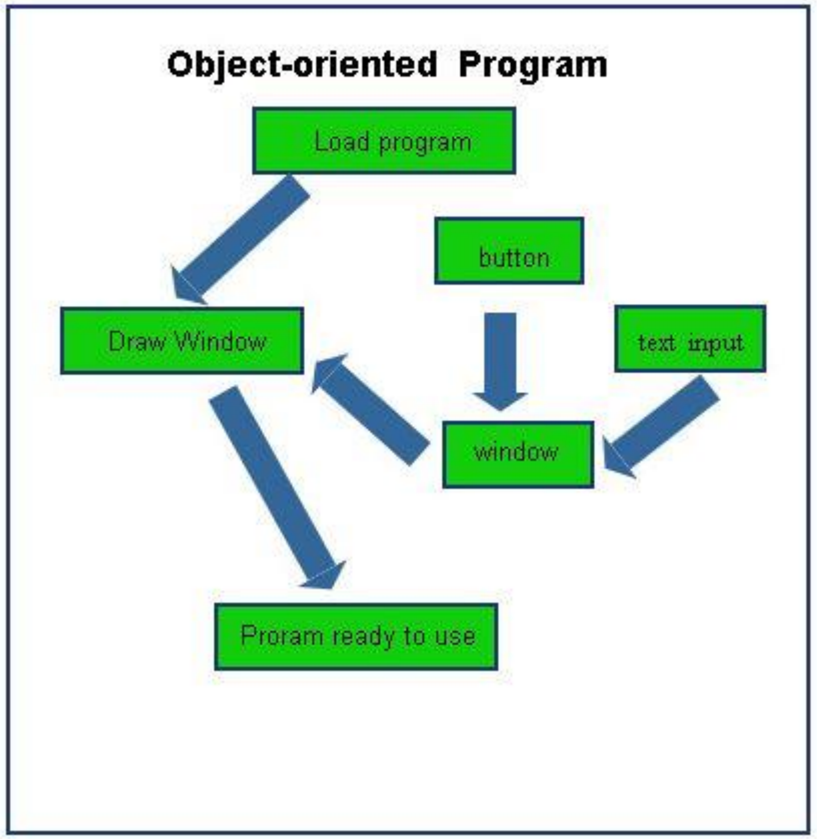


- **OOP** - reasoning about a program as a set of objects rather than a set of actions
    - **‘noun-verb’** vs **‘verb-noun**’ paradigm (aka **windows** vs **cmd**)
- **Object** – a programming entity that contains state (data) and behavior (methods)
- **State** - a set of values stored in an object.
- **Behavior** – a set of actions an object can perform, often reporting or modifying its internal state.
- **Client** (Client Code) – code that interacts with a class or objects of that class

OOP from 10,000 feet : visually



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages



Examples

- Time (12:59 am)
- Date (1/1/2014)
- Student (id, name, GPA, major,…)
- Color (r, g, b, alpha)
- Fraction (3/4)

- Can you think of more examples?

1 minutes: 2 examples

Additional Examples

- A Point
  - Two points compose a line, or vertex, or ray
- Shape
  - A Square is a Shape, a Circle is a Shape
- List, Stack, Queue
  - Data structures in 1 dimension

Classes vs Objects

- One is a **text file**; the other a chunk of **RAM**
  - One is persistent
- One is the **cookie-cutter**, the other the **cookies**
- Generally used interchangeably in the literature, which is unfortunate.
  - *Java doesn’t help by naming it’s top-level class “Object”.*

Point2D Class

```
Class Point2D {
    //data section
    int x;
    int y;

    //methods section
    void resetToOrigin() {}
    void move(int dx, int dy) {}
    double getDistanceFrom(Point2D other) {}
}
```

DateV1.0

```
class Date
{
    //what data?



    //what methods?


}
```

3 mins:

Define types of data and what do you need to do with it

CLASS DEFINITIONS (Savitch 4.1)

- Below slides are a quick review of the DEFINITIONS
- I will only emphasize key points
- Refer to **Savitch Chapter 4.1** for more details