Notes: OOP

Object Oriented Programming (OOP)

* **OOP**: programs that perform their behaviour as interactions between objects
* Objects group together related variables
* An **object** is an entity that combines **state** and **behavior**
  + An object is a blueprint for a new data type
  + An object is not executable (it does not contain a main method)
* A created object (using the new keyword) is an **instance** of a class
* A **client program**, is a program that uses objects (you have already been writing client programs)

State

* An object's state is defined by the collection of things that it can remember (fields/instance variables)
* **Field**: A variable inside an object that remembers something
  + Each object has its own copy of each field
  + To access use <objectinstance>.<field>
  + To modify use <objectinstance>.<field> = <value>
* Not every variable should be a field, only the things the object needs to remember

Encapsulation

* Hiding implementation details of an object from its clients
* Implemented with the private keyword before fields; which stops code outside of the class from being able to access it
* If you then want to give access, you add an accessor and mutator method
* Encapsulation provides **abstraction** between an object and its clients
* Encapsulation protects objects from unwanted access by clients

Behavior

* **Instance method**: A method inside an object that operates on that object
* **Implicit parameter**: The object on which an instance method is called; can be referred to with the this keyword
* **Constructor**: a special method that has the same name as the class and is used to create an Object instance from the class's blueprint in another program
  + Must be public, same name as the class, and should initialize all instance variables
* **Accessor**: a method that provides information about the state of an object; giving clients "read only" access to the object's fields
  + Generally uses the keyword "get" on the method name, takes no parameters, and returns the value of an instance variable
* **Mutator**: a method that modifies the object's internal state; giving clients both read and write access
  + Generally used the keyword "set" on the method name, takes a parameter for the new value of an instance variable in the object, and reassigns an instance variable to the passed parameter (returns void)
* **toString()**: a method that allows you to print out the contents of the Object's state

More on Constructors

* The constructor's name needs to be the same as the name of the class itself
* Initializes the state of new objects
* Runs when the client uses the new keyword
* Does not specify a return type (no void, no other data type, just nothing)
* Implicitly returns the new object that was created
* Every class needs a constructor, but if you do not define a constructor then Java gives it a default constructor with no parameters and sets all fields to zero
* A class can have multiple constructors, but they each have to have a unique set of parameters
  + If you do have multiple constructors it is possible for one constructor to call another constructor

More on toString()

* By convention, when you print an object Java will always call the toString() method of that class
  + This is because every single object inherits from the Object class in Java, and within that Object class there is a toString() method
* By default, Java includes a toString() method that returns a String containing the memory address of that object
* You can override the toString() method to customize the String that it returns

Example Code

/\*

This is an example of what happens inside java.awt.Point

https://docs.oracle.com/javase/8/docs/api/index.html?java/awt/Point.html

\*/

// procedural programming: a program as a sequence of commands

// object oriented programming: a group of interacting Objects

// class: a blueprint for creating an object;

// defines an object's state and behavior

// state: things the Object remembers (fields)

// behavior: things the Object can do (methods)

public class Point {

// fields/instance variables: (combined make up the object's state)

// should be private and declared (but not initialized)

int x;

int y;

// constructors: allow you to create an Object from this class

// no return type, same name as the class, initalize all fields

public Point() {

x = 0;

y = 0;

}

public Point(int startX, int startY) {

x = startX;

y = startY;

}

// accessors: allow you to access a field of an Object's state

// use "get", return a field's value, no params

public int getX() {

return x;

}

public int getY() {

return y;

}

// mutators: allow you to change a field of an Object's state

// use "set", take a parameter for new value, changes field variable

public void setX(int newX) {

x = newX;

}

// OTHER BEHAVIOR: other methods that don't follow the structure

// of accessors and mutators

// Moves this point to the specified location in the (x,y) coordinate plane.

public void move(int x, int y) {

this.x = x; //this.setX(x) or setX(x)

this.y = y;

}

// Translates this point, at location (x,y), by dx along the x axis

// and dy along the y axis so that it now represents the point (x+dx,y+dy).

public void translate(int dx, int dy) {

x += dx; // this.x = this.x + dx;

y += dy;

}

// toString: allows you to print out the state of an Object

// must return a string, should NOT have System.out.println here

public String toString() {

return "(" + x + "," + y + ")";

}

}