

CS 2113

Software Engineering

Java Module 2:
Object Oriented Programming

`https://github.com/GWU-CSCI2113/java2-oop.git`

Java Module 1

- Your code goes in your README file
- Use proper markdown formatting!

```
```java
```

Markdown syntax to start a code block with java syntax highlighting!

```
public class Hello{
// ...

}
```
```

This Time

- Java Objects and Classes
 - Objected oriented vs Procedural programming
 - Access control
 - Inheritance
- Basic UML Diagrams
 - Representing code visually
 - Class Diagrams

The Java docs

- The main documentation of the Java language is in the form of "javadocs"
 - Created from informative comments in the actual source code
- Java 10
 - <https://docs.oracle.com/javase/10/docs/api/overview-summary.html>

Practice reading these!

Textbook Reading

- Read Chapters 4, 5, and 7
 - Objects and inheritance
- Do this before the next lab!

Procedural vs Object Oriented

- Programming is about **functions** and **data**
 - In C (a *procedural* language) those were kept separate:

```
struct employee {  
    char* name;  
    double salary;  
};  
  
double getTaxRate(struct employee* e) {...}
```

- Procedural programming focuses on **what needs to be done**
- Object oriented programming focuses on **the data** that must be manipulated

Nouns vs Verbs

- Procedural is about actions (verbs)
- O.O. is about things (**nouns**)
- A simple program:

Write a program that asks for a **number** and then prints its square root.

- A more realistic example:

A *voice mail system* records calls to multiple *mailboxes*. The system records each *message*, the *caller's number*, and the *time of the call*. The *owner* of a mailbox can play back saved messages.

Benefits of O.O.P.

- Focuses on **Data** and **Relationships**
- Break functionality of program down into interacting objects
 - Divide and conquer
- Clearly defined interfaces between components
 - Fewer bugs from unexpected interactions
 - Ask an object to modify itself
- Supports hierarchies of objects

Objects and Classes in Java

- In Java:

- A class is type of object
- All objects have a class
- Primitives (int, float, etc) and functions are not objects


```
public class Hello {  
    public static void main () {  
        System.out.println("hi.");  
    }  
}
```

- Classes contain data and functions

- An object instantiates that data

- Class Constructor

- Used to create new objects
- Initialize variables
- Must call **super(...)** if a subclass



```
public class Car {  
    public int x;  
    public int y;  
  
    public Car(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
}
```

Creating new Objects

- To use an object we need a reference to it
- Use the **new** command to instantiate an object
 - Reserves memory on the Heap for that object class
- Each new object gets its own chunk of memory
 - But they share functions and static class variables

```
public static void main(...)  
{  
    Car chevy;  
    Car honda;  
    honda = new Car(10, 20);  
}
```

Stack

| | |
|-------|-----------|
| chevy | null |
| honda | 0x1423000 |

Heap

| | | |
|-----------|---|----|
| 0x1423000 | x | 10 |
| 0x1423004 | y | 20 |



Racing

- Get the code and run it (press **ctrl-c** to quit):

```
cd java2-oop/proc  
javac SimpleTrack.java  
java -cp . SimpleTrack
```

- Simple changes
 - Green car
 - Different car symbol
 - Faster car
- Harder changes:
 - Have two cars instead of one
 - (Don't add any new classes yet)

static Members

- **static** Functions and Data are not specific to any one object instance
 - One copy is kept for the entire class
- You can access **static** members with just the class name
 - Do not need a reference to a specific object
- Useful for globally available data or functions that do not depend on any one object instance

Class
name

```
Math.random() // static method in Math class  
Math.PI // static data object in Math class
```

Object
name

```
Car myCar = new Car(10,10);  
myCar.x // instance variable
```

Uses of `static`

- #1: Great for classes that cannot be instantiated:
 - **Math** class
 - Provides general utility methods, doesn't have any data that must be unique for each object instance. You never do "new Math()"
- #2: Useful for sharing a single variable for all instances of a class
 - **carsBuilt** counter
 - Isn't unique to each car
 - Shouldn't need a reference to a car object to find out how many have been made
- Just don't abuse it

```
public class Car
{
    public static int carsBuilt;
    public String license;
    public Car(){
        carsBuilt++;
        // ....
    }
}
```

```
Car c = new Car();
System.out.println(c.license);
System.out.println(Car.carsBuilt);
```

Modularizing the code

- **Terminal** class: constants related to printing colors
- **Car** class: a "V" drawn to the screen that randomly moves back and forth. Starts at random position.
 - Data?
 - Functions?
- **RaceTrack** class: instantiates a car object and has the main run loop of the program

Work in java2-oop/oop

Loop forever...

update the car's position

for each horizontal position in the line...

if the car is at that position, draw it

else draw an empty space

print newline character and reset the color

sleep for 30 milliseconds

Lots of objects

- How do we make an array of objects?

```
Car cars[10];  
  
for(int i=0; i < cars.length; i++) {  
    cars[i].move();  
}
```

Lots of objects

- How do we make an array of objects?

```
Car cars[] = new Car[3];

for(int i=0; i < cars.length; i++) {
    cars[i] = new Car();
}

for(Car c: cars) { // fancy array looping
    c.move();
}
```

- Note: We can't use the fancy loop when allocating the cars... **why?**

Lots of cars!

- Modify your program so it can create 3 cars at random positions and show them driving around
- Too easy? Try these:
 - Each car can be a different color
 - Make the left and right sides of the race track dynamically change width and don't let any cars go beyond them
 - If two cars collide, they should explode (i.e., display X and stop printing those two cars from then on)
 - Make the program stop when only one car is left

Inheritance

- Classes can be defined in a hierarchy
- The child (subclass) inherits data and functionality from its parent (superclass)
- Use the **extends** keyword in java

```
public class RocketCar extends Car {  
    public int numEngines; // define new class members  
  
    public RocketCar(int x, int y, int n) {  
        super(x,y); // must call parent's constructor  
        numEngines = n;  
    }  
    public void draw()  
    {  
        drawSmokeTrail();  
        super.draw(); // optionally call parent version  
    }  
}
```

Functions

- Inherited from parent class
 - Can be replaced fully, or can call **super.funcname()** to combine parent and child functionality
- Supports multiple declarations with different parameters

```
public void draw(Color c)
{
    // draw with specific color
}
public void draw()
{
    // this code is unnecessary
    super.draw()
}
```

Vehicle Types

- Extend your program to define multiple types of vehicles or objects with inheritance
- Some options:
 - Wild cars move more erratically than regular race cars
 - Some cars look different from others
 - Trees only appear on some lines
 - Walls are at the edges of the track and move less erratically than cars. If a car hits a wall it crashes
 - ...something more creative that you think of...

Limiting Visibility

- Goal of OOP: compartmentalize data and function
- Classes can be:
 - **public** - visible everywhere
 - (no keyword) - visible within package
- Data and functions can be:
 - **public** - callable, readable/writable by anyone
 - **private** - only accessible within the same class
 - **protected** - accessible in the class, subclass, and package
 - (no keyword) - accessible within the package

Visibility

- Answer question 1 on the worksheet.
- Data and functions can be:
 - **public** - callable, readable/writable by anyone
 - **private** - only accessible within the same class
 - **protected** - accessible in the class, subclass, and package
 - (no keyword) - accessible within the package

Summary

- Java emphasizes Object Oriented Programming
- We use OOP to write better structured code
 - A correct program is always better than a fast buggy one
- Think about how you organize classes
 - Use private variables and expose clean interfaces
 - Use inheritance to reuse code
- UML can help plan your code