Introduction to Inheritance

In this lecture we will cover:

- Four pillars of object oriented programming
- Basic inheritance
- The Java Object class

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:
  - Computer science has a number of concentrations.
    - General concentration is the most flexible, and even allows students to double major or minor pretty easily.
    - Software Engineering
    - Computing Systems
    - Human Centered Computing
    - Networks and Security
    - Artificial Intelligence
    - Computer Science Education.
  - Minors:
    - Minor in Computer Science - choose your own adventure minor
    - Minor in Machine Learning - popular with stats/math, and engineering
    - Minor in Bioinformatics - Biology + Computer Science

Four Pillars of Object Oriented Programming

- Abstraction
  - You have practiced this already by building 'general' classes/types
  - You can reuse these types in a variety of different ways
- Encapsulation

- You have practiced this by
  - Making classes wrap/focus on a single idea
  - Controlled entry points into that class (public / private control)
- Inheritance
  - The ability to create an **is a** relationship between classes
  - Classes inherit from other classes, gaining their properties
- Polymorphism
  - Intrinsically connected to Inheritance
  - We will cover this in a future lecture in Unit 3

These are nice technical interview questions, but not something we talk about daily. Understanding them helps you understand the "why" on how java is designed (every class can be an object, and one class per file embodies encapsulation!)

Inheritance

- Allows you to build relationships
  - Between a more general class (called superclass also parent class)
  - And a more specialized class (called subclass or child class)
- Or another way to look at it
  - By looking at types/ideas as generalized to more specific
  - You can write your code **once** and let the more specialized classes use the generalized code

Example:

- Boxes come in different shapes and sizes
  - but a *cube* is a specific box with the sides the same in all dimensions
  - Let's look at some code that represents it!

In [6]:
```java
public class Box {
    protected int width;
    protected int height;
    protected int length;
    public int getArea() { return width*height*length;}
    public String toString() {
        return String.format("Width: %d, Height: %d, Length:%d", width, height, lengtl
    }
    public Box(int width, int height, int length) {
        this.width = width;
        this.height = height;
        this.length = length;
    }
}

public class CubeBox extends Box {
    public CubeBox(int side) {
        super(side, side, side);
    }
    public String toString() {
        return String.format("Side Size: %d", width);
    }
}

Box bx = new Box(10, 12, 7);
CubeBox cb = new CubeBox(5);
System.out.println(bx + " and the area is " + bx.getArea());
System.out.println(cb + " and the area is " + cb.getArea());
```

```
Width: 10, Height: 12, Length:7 and the area is 840
Side Size: 5 and the area is 125
```

> Discussion
> Discuss at your table what you notice about the code? Some is talked about in the reading, can you help each other understand it? We will talk about it in a moment, but for now take a moment to explain the code to others at the table.

Inheritance Definitions

- **extends** keyword says
  - Inherit the methods and properties (variables) from the superclass Box
  - Critical word to have
- CubeBox
  - has both the method created in CubeBox *plus* the methods in Box
    - getArea() is in both, functions the same in both!
- **super**
  - Is you reference the superclass (parent).
  - super(...) - is calling the parent classes constructor
  - Not required, but common!
- **protected**
  - New scope (you already know private and public)
  - protected scope subclasses have direct access to, but only the current class and subclasses.
- toString()?
  - both classes have a toString() method
    - which means that specific classes version is called
    - this is called **Overwriting**

In Class Activity

You will find in canvas/zybooks/github the class `Employee.java`. It simply contains an employee ID and name.

You goal is to write a new class (notice it is completely blank in zybooks) called `SoftwareEngineer.java`

- SoftwareEngineer
  - sets the Employee pay scale to be 75,000-250,000
  - has a jobcode associated with them
  - needs a getJobCode() and setJobCode() method
  - needs a constructor of the format `SoftwareEngineer(int id, String name)`
- Make sure to write some tests, so you get something useful to print

In [13]:
```java
public class Employee {
    private final int pay_low;
    private final int pay_high;
    private String name;
    private final int employee_id;

    public int getLowPay() { return pay_low;}
    public int getMaxPay() { return pay_high;}
    public int getID() { return employee_id;}
    public String getName() { return name;}
    public void setName(String name) { this.name = name;}
    public String toString() { return String.format("Name: %s, ID: %d", name, employee
    public Employee(int id, String name, int pay_low, int pay_high) {
        employee_id = id;
        this.pay_low = pay_low;
        this.pay_high = pay_high;
        this.name = name;
    }
}



public class SoftwareEngineer extends Employee {
    private String jobCode;

    public SoftwareEngineer(int id, String name) {
        super(id, name, 75000, 250000);
    }
    public void setJobCode(String code) { jobCode = code;}
    public String getJobCode() { return jobCode;}

}

Employee hr = new Employee(10, "Lucky Lou", 40000, 60000);
SoftwareEngineer se = new SoftwareEngineer(42, "Alice Wonder");
se.setJobCode("REDQUEEN01");

System.out.println(hr);
System.out.println(se);
```

```
Name: Lucky Lou, ID: 10
Name: Alice Wonder, ID: 42
```

Object class

All classes inherit from Object.java

This is assumed, and even if you don't put *extends Object*, it is already there.

- Another way to put it, Objects are the cells of Java, everything is based on them.

Why is this beneficial?

- .toString()
  - This is implemented in object as returning the memory address
  - As all classes have it, other objects (ArrayList) can use it to print!
  - When you **overwrite** it, it calls your version of toString()
  - You have already been doing this
- .equals(Object)
  - By default compares the memory address
  - However, String overwrites it to compare the characters in the string
  - You can overwrite it in your classes to give meaning on comparing them

There are a lot of other methods Object provides to all classes you write, but those are the most common to overwrite (getHash() is another one).

In Class Activity Two

- Add a toString method to SoftwareEngineer
- It should print out

```
Name: name, ID: id, JobCode: jobcode
```

In [14]:
```
public class SoftwareEngineer extends Employee {
    private String jobCode;

    public SoftwareEngineer(int id, String name) {
        super(id, name, 75000, 250000);
    }
    public void setJobCode(String code) { jobCode = code;}
    public String getJobCode() { return jobCode;}

    public String toString() { return super.toString() + ", JobCode: " + getJobCode();

}

Employee hr = new Employee(10, "Lucky Lou", 40000, 60000);
SoftwareEngineer se = new SoftwareEngineer(42, "Alice Wonder");
se.setJobCode("REDQUEEN01");

System.out.println(hr);
System.out.println(se);
```

```
Name: Lucky Lou, ID: 10
Name: Alice Wonder, ID: 42, JobCode: REDQUEEN01
```

Overview

General rules for inheritance

- Use it. The *is a* relationship is valuable
    - Good inheritance saves you time writing code
    - Bad inheritance increases work loads
- Really focus on the idea and breaking up the idea
- If you can, call the superclass methods with slight modifications
    - This helps incase the superclass changes implementations
- Right now the primary reason for inheritance?
    - Keeping your code DRY
    - In the future, we will learn about polymorphism which doubles the power of inheritance.