INFO 5100
Application Engineering and Development

Week 4

Agenda

- Review Class
- Inheritance

Review Class

```java
public class Dog {
   private String name;
   public Dog() {
       this.name = "no name";
   }
   public Dog(String name) {
       this.name = name;
   }
   public String getName() {
       return this.name;
   }
   public void bark() {
     System.out.println("my name is: " + this.name);   }
```

```java
@Override
public boolean equals(Object o) {
    if (o == this) return true;
    if (!(o instanceof Dog)) return false;
    Dog d = (Dog)o;
    return this.name.equals(d.name);
}

@Override
public String toString() {
    return this.name;
}
```

Class

- Properties of an Object are determined by its instance variables
    - Also known as Fields or Member Variables
- Behavior of an Object are determined by its instance methods
    - Constructors are special

Inheritance

- Imagine you are tasked to implement an Employee class
  - The Employee has some basic properties
  - The Employee should get paid

Employee

```java
public class Employee {
    // properties of an employee
    private String firstName;
    private String lastName;
    private int age;
    private int salary;
    public void work() {
        System.out.println("i did some work");
    }
    public void getPaid(int amount) {
        this.salary += amount;
    }
}
```

Inheritance

- The requirement changed to support different types of Employees
    - Salaried Employee
    - Hourly Employee
- Salaried Employee can enjoy things like healthcare, more PTO, and get paid twice a month
- Hourly Employee does not have healthcare, does not have PTO, and get paid daily

- How to achieve this?

Non-Inheritance Approaches

- Copy and Paste all the employee code and create two classes
  - SalariedEmployee
  - HourlyEmployee
- Keep the single class Employee
  - Add a status flag
  - Using switch or if statements in every method

Non-Inheritance Drawbacks

- There could potentially be huge amount of Employee types
- There could be bug in initial implementation of Employee class before Copy and Pasting
- How do you manage Employees together? Instead of two collections

- Maintainability and Extensibility are poor

Abstraction

- Models relationship accurately
- Without forcing the user to keep track of more than necessary

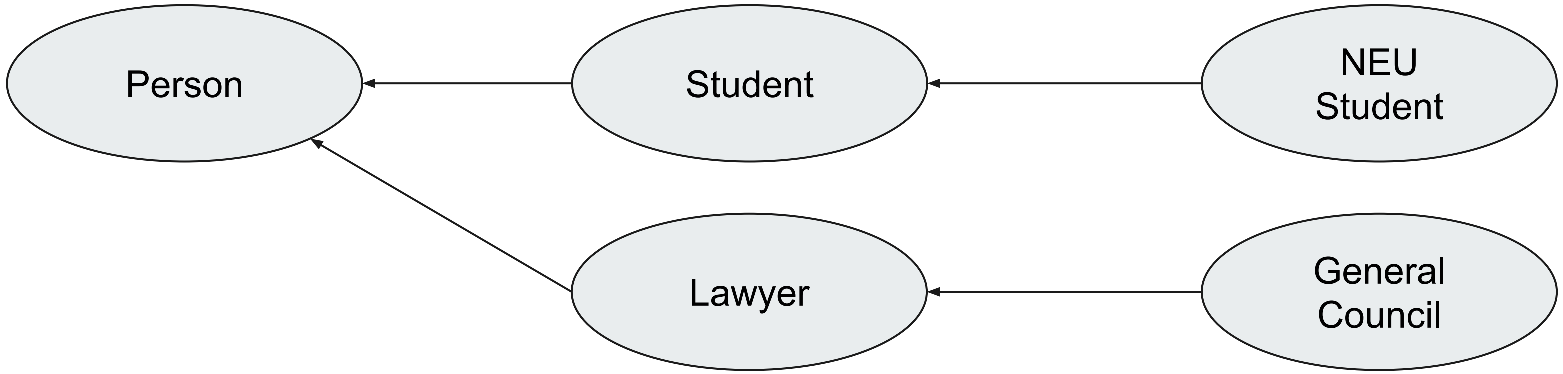
HourlyEmployee

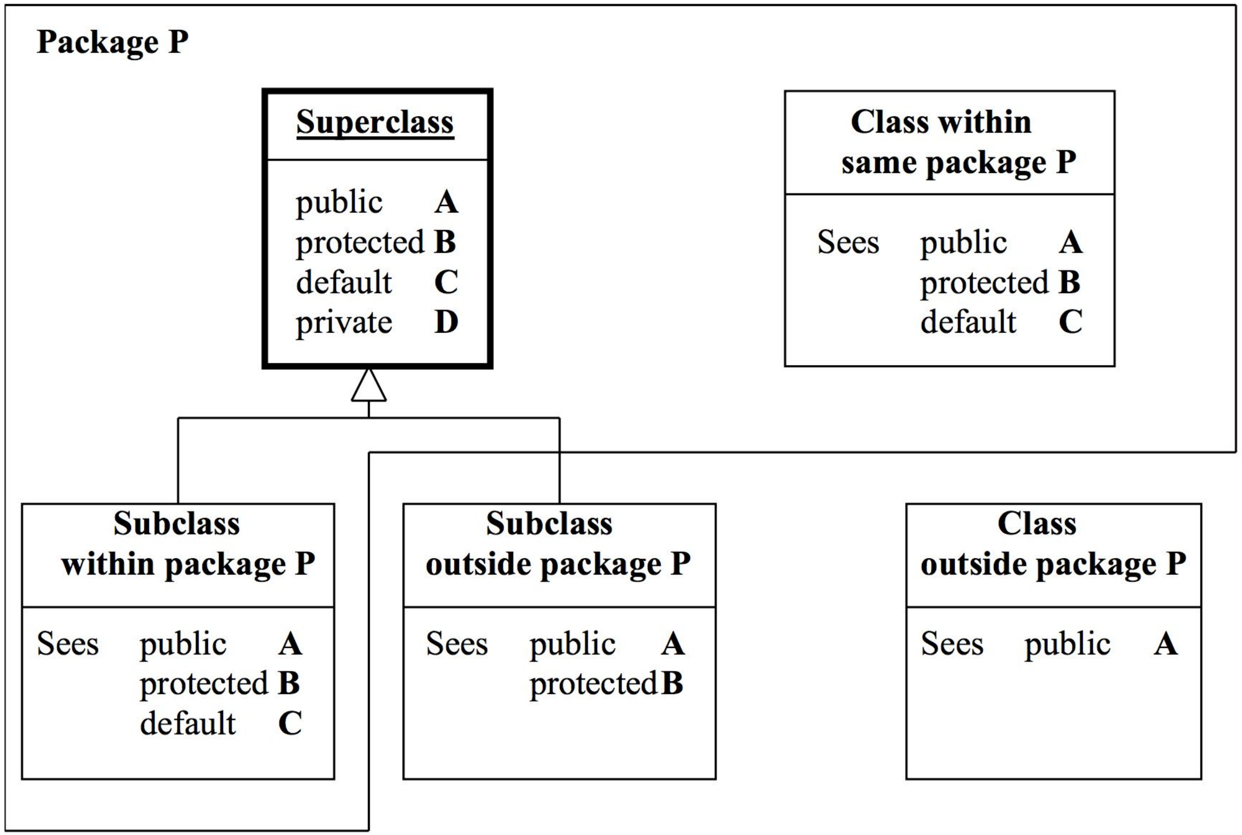
```java
public class HourlyEmployee extends Employee {

    public void withdrawSalary() {
        // logic to allow withdraw of salary daily
    }

    public static void main(String[] args) {
        HourlyEmployee he = new HourlyEmployee();
        he.work();
    }
}
```

Inheritance

- Use extends keyword to declare inheritance relationship
    - HourlyEmployee inherits from Employee
    - Employee is a base/superclass for HourlyEmployee
    - HourlyEmployee is a derived/subclass of Employee
- Subclass could have instance variables that are not defined in superclass
- Subclass could have instance methods that are not defined in superclass
- Subclass do not need to redeclare all the instance variables and methods in superclass
- Subclass inherits properties/behaviors except those marked Private

Inheritance Hierarchy





Access Modifier Visibility

Inheritance

- Subclass optionally can override a method inherited from a superclass
    - In previous class we saw example with toString and equals
- Subclass can use **super** keyword to invoke superclass method

Inheritance

- You cannot **extend** multiple classes in Java
- Example
  - A is the super class
  - B extends A, overrides method A.doSomething()
  - C extends A, overrides method A.doSomething()
  - D extends B and C, but do not override any method
  - Now we do D.doSomething(), which version to call?

Types in Java

- Type of the variable and the value must match*
- In some cases, Java performs auto **type casting**
  - Numeric values might loss precision
  - Sometimes during operations

Polymorphism

```
Employee e1 = new String("i am a string");
Employee e2 = new HourlyEmployee();
HourlyEmployee e3 = new Employee();
```

Polymorphism

- At runtime select appropriate behavior based on the reference
- It is possible to treat superclass and subclass similarly
    - Objects of all types derived from a common superclass can all be treated as Objects of superclass type

Polymorphism

- The is-a rule
  - HourlyEmployees are always Employee
  - Employees are not always HourlyEmployee
- The variable type determines what kind of properties/behaviors you can access
- You can manually perform typecasting to get access to subclass properties/behaviors
  - Could generate runtime error
- Use **instanceof** operation to perform check on runtime before typecasting

Polymorphism

- Use **final** keyword to prevent overriding and extending
    - final method cannot be overridden in a subclass
    - final class cannot be extended
        - All methods in the class became implicitly final

Abstract Class

- With levels of inheritance it is hard to control the usage of superclasses
- **abstract** can be used to enforce subclass must implement certain methods

Abstract Class

```
public abstract class AbstractEmployee {
   public String name;

   public abstract void work();
}
```

Abstract Class

```java
public class ConcretEmployee extends AbstractEmployee {

    @Override
    public void work() {
        System.out.println("doing work");
    }

    public static void main(String[] args) {
        // AbstractEmployee ae = new AbstractEmployee();
        ConcretEmployee ce = new ConcretEmployee();
        ce.work();
    }
}
```

Abstract Class

- Declared with keyword **abstract**
- Cannot be instantiated with **new** keyword
- A class with at least one abstract method must be defined as abstract class

Interface

- More abstract than abstract class
- Most of time do not have any concret/default implementation of functions
- Only public abstract method declarations and public constants
- Not instantiable

- Can be used for multi-inheritance

Interface

```java
public interface Worker {


    public void work();


}
```

Interface

```
public class ConcretEmployeeV2 implements Worker, Comparable<ConcretEmployeeV2> {
    public String name;
    public void work() {
        System.out.println("do some work");
    }
    public int compareTo(ConcretEmployeeV2 c) {
        return this.name.compareTo(c.name);
    }
    public static void main(String[] args) {
        ConcretEmployeeV2 c = new ConcretEmployeeV2();
        c.work();
    }
}
```