Inheritance

Chapter 8 in Savitch
Chapter 8 in Lewis

The software crisis

- **software engineering**: The practice of conceptualizing, designing, developing, documenting, and testing large-scale computer programs.
- Large-scale projects face many issues:
  - getting many programmers to work together
  - getting code finished on time
  - avoiding redundant code
  - finding and fixing bugs
  - maintaining, improving, and reusing existing code
- **code reuse**: The practice of writing program code once and using it in many contexts.

Example

- You have been tasked with writing a program that handles pay for the employees of a non-profit organization.
- The organization has several types of employees on staff:
  - Full-time employees
  - Hourly workers
  - Volunteers
  - Executives

Design

- Need class/classes that handle employee pay (should also store employee info such as name, phone #, address).
- Possible choices:
  - A single Employee class that knows how to handle different types of employees
  - A separate class for each type of employee.
- What are the advantages/disadvantages of each design?

Example

- Paying an employee:
  - Full-time employees – have a monthly pay
  - Hourly workers – hourly wages + hours worked
  - Volunteers – no pay
  - Executives – receive bonuses

Design

- All types of staff members need to have some basic functionality – capture that in a class called **StaffMember**
Design

All types of staff members need to have some basic functionality – capture that in a class called `StaffMember`

```java
class StaffMember {
    private String name;
    private String address;
    private String phone;

    public StaffMember(String eName, String eAddress, String ePhone) {
        name = eName;
        address = eAddress;
        phone = ePhone;
    }
    // not shown: getters and setters
}
```

Code re-use

- We’d like to be able to do the following:

  ```java
  public class Employee {
      private double payRate;
      public double pay() {
          return payRate;
      }
      // not shown: getters and setters
  }
  ```

- All this without explicitly copying any code!

Inheritance

- **Inheritance**: A way to form new classes based on existing classes, taking on their attributes/behavior.
  - A way to group related classes
  - A way to share code between two or more classes
- A class extends another by absorbing its state and behavior.
  - **superclass**: The parent class that is being extended.
  - **subclass**: The child class that extends the superclass and inherits its behavior.
  - The subclass receives a copy of every field and method from its superclass.

Inheritance syntax

- Creating a subclass, general syntax:
  ```java
  public class <name> extends <superclass name> {
      ....
  }
  ```
- Example:
  ```java
  public class Employee extends StaffMember {
      ....
  }
  ```
- By extending `StaffMember`, each `Employee` object now:
  - has `get/setName`, `get/setAddress`, `get/setPhone` automatically
  - can be treated as an `StaffMember` by any other code (seen later)
  - (e.g. an `Employee` could be stored in a variable of type `StaffMember` or stored as an element of an `StaffMember[]`)

Extends/protected/super

```java
public class Employee extends StaffMember {
    protected String socialSecurityNumber;
    protected double payRate;

    public Employee(String eName, String eAddress, String ePhone, String socSecNumber, double rate){
        super(eName, eAddress, ePhone);
        socialSecurityNumber = socSecNumber;
        payRate = rate;
    }

    public double pay(){
        return payRate;
    }
}
```
Overriding methods

- **override**: To write a new version of a method in a subclass that replaces the superclass’s version.
  - There is no special syntax for overriding. To override a superclass method, just write a new version of it in the subclass. This will replace the inherited version.
  - Example:
    ```java
    public class Hourly extends Employee {
        // overrides the pay method in Employee class
        public double pay() {
            double payment = payRate * hoursWorked;
            hoursWorked = 0;
            return payment;
        }
    }
    ```

Calling overridden methods

- A subclass can call an overridden method with the `super` keyword.
- Calling an overridden method, syntax:
  ```java
  super. <method name> (<parameter(s)>)
  ```
  - Example:
    ```java
    public class Executive extends Employee {
        public double pay() {
            double payment = super.pay() + bonus;
            bonus = 0;
            return payment;
        }
    }
    ```