# Inheritance: Overriding

## CPSC 233: Introduction to Computer Science for Computer Science Majors II
## Winter 2022

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

**Wednesday, 10 November 2021**



UNIVERSITY OF CALGARY

# Overriding Methods

- A child class can *override* (i.e. re-define) the definition of an inherited method
    - if the parent's code not applicable, write specific code for the child
    - If a method is declared with the `final` modifier, it cannot be overridden
- The new method must have the same signature as the parent's method

# final

Prevent you from overriding a method

Compile time error if we attempt to

```
public class Person {

    protected String name;
    protected int id;

    public final String getName() {
        return name;
    }
}

public class Student extends Person{

    ArrayList<Session> classes;

    public String getName() {
        return name;
    }
}
```

# Example - Overriding Methods

What if we wanted to print details about Student/Staff/Person?

1. In Person?
   - We don’t know anything about details of Staff or Person.
2. In both Person and Staff?
   - We will have to duplicate name/id printing.
3. Some in all 3?
   - Use power of **super.** and method **override**

# Interacting with super class

You can override a method from your parent class by using the same variable or method name

(@Override is optional keyword put over method)

```
public class Person{
    String name;
    int id;
    public String getName(){
        return name;
    }
}
public class Staff extends Person{
    Staff boss;
    @override
    public String getName(){
        return "Staff: " + name;
    }
}
```

# Interacting with super class

You can access super classes methods/variables if you've re-used the name using **super.**

- If not overriden, then can just use name
- Works like **this.** does in a regular class

```
public class Person{
    String name;
    int id;
    public String getName(){
        return name;
    }
}
public class Staff extends Person{
    Staff boss;
    @override
    public String getName(){
        return "Staff: " + super.getName();
    }
}
```

# Example - Overriding Methods

```
public class Person {
    private String name;
    private int id;

    public String printVersion() {
        return String.format("%s(%s)", name, id);
    }
}

public class Student extends Person {
    private ArrayList<Session> classes;

    @Override
    public String printVersion() {
        return String.format("%s Classes->%s", super.printVersion(), classes);
    }
}

public class Staff extends Person {
    private Staff boss;

    @Override
    public String printVersion() {
        return String.format("%s Boss->%s", super.printVersion(), boss);
    }
}
```

# Overloading vs. Overriding

**Overloading**

- multiple methods with the same name in the same class, but with different signatures
- lets you define a similar operation in different ways for different parameters

e.g. class constructors:

```
... = new Student();
... = new Student ("123456789", "Smith, John");
```

Triangle class:

```
t.setSide(10)    // all 3 sides same
t.setSide(3, 4, 5)   // all different
```

# Overloading vs. Overriding

**Overriding**

- deals with two methods **in different classes** (one in a parent class and one in a child class), that have the **same signature**
- lets you define a similar operation in different ways for different object types

# Onward to … Hierarchies

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY