

# Inheritance: Creating

---

**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**

*Copyright © 2021*



# Inheritance

---

- a fundamental object-oriented design technique used to create and organize reusable classes

# The programming inefficiency?

---

```
public class Faculty{
    String name;
    int id;
    ArrayList<Class> lectures;
    public String getName(){
        return name;
    }
}
```

```
public class Student{
    String name;
    int id;
    ArrayList<Class> classes;
    public String getName(){
        return name;
    }
}
```

```
public class Staff{
    String name;
    int id;
    Staff boss;
    public String getName(){
        return name;
    }
}
```

# The programming inefficiency?

---

## Code Repetition

- These 3 classes are really similar in function
- Shared
  - Storage variables
  - Methods to access (and change) data

```
public class Faculty{
    String name;
    int id;
    ArrayList<Class> lectures;
    public String getName(){
        return name;
    }
}
public class Student{
    String name;
    int id;
    ArrayList<Class> classes;
    public String getName(){
        return name;
    }
}
public class Staff{
    String name;
    int id;
    Staff boss;
    public String getName(){
        return name;
    }
}
```

# The programming inefficiency?

---


## Code Repetition

- These 3 classes are really similar in function
- Shared
  - Storage variables
  - Methods to access (and change) data
- If we change what is a valid institutional id
  - We'll have to change code in all 3

```
public class Faculty{
    String name;
    int id;
    ArrayList<Class> lectures;
    public String getName(){
        return name;
    }
}
public class Student{
    String name;
    int id;
    ArrayList<Class> classes;
    public String getName(){
        return name;
    }
}
public class Staff{
    String name;
    int id;
    Staff boss;
    public String getName(){
        return name;
    }
}
```

# Outline

---

- 
- Creating Subclasses
  - Overriding Methods
  - Class Hierarchies
  - Designing for Inheritance

# The Goal

---

## Code re-use through hierarchies

We'd rather write code to store the **common**

- **State** (variables)
- **Behaviour** (methods)

**Once** and access this behaviour from all 3 classes that share it

# Solution

---

## Code re-use through hierarchies

We'd rather write code to store the **common**

- **State** (variables)
- **Behaviour** (methods)

**Once** and access this behaviour from all 3 classes that share it

```
public class Person{  
    String name;  
    int id;  
    public String getName () {  
        return name;  
    }  
}
```



# Solution

---

## Code re-use through hierarchies

We can inherit this code using **keyword -> extends**

Ex. <class\_1>, <class\_2> are two Java classes

```
public class <class_1> extends <class_2> {  
}
```

# The Goal

---

## Code re-use through hierarchies

```
public class Person{
    String name;
    int id;
    public String getName(){
        return name;
    }
}
```

```
public class Faculty extends Person{
    ArrayList<Class> lectures;
}
public class Student extends Person{
    ArrayList<Class> classes;
}
public class Staff extends Person{
    Staff boss;
}
```

# The Goal

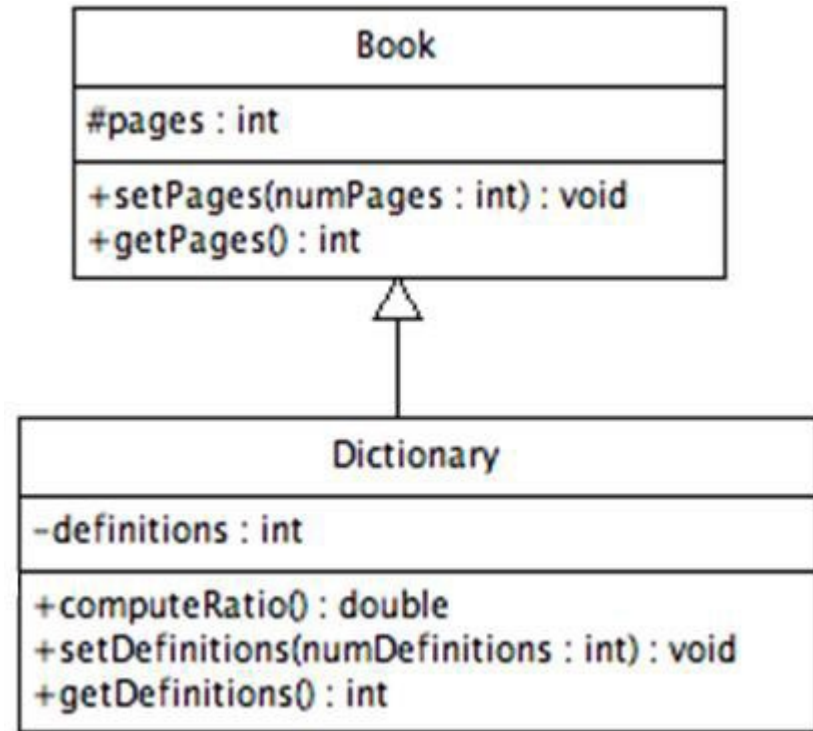
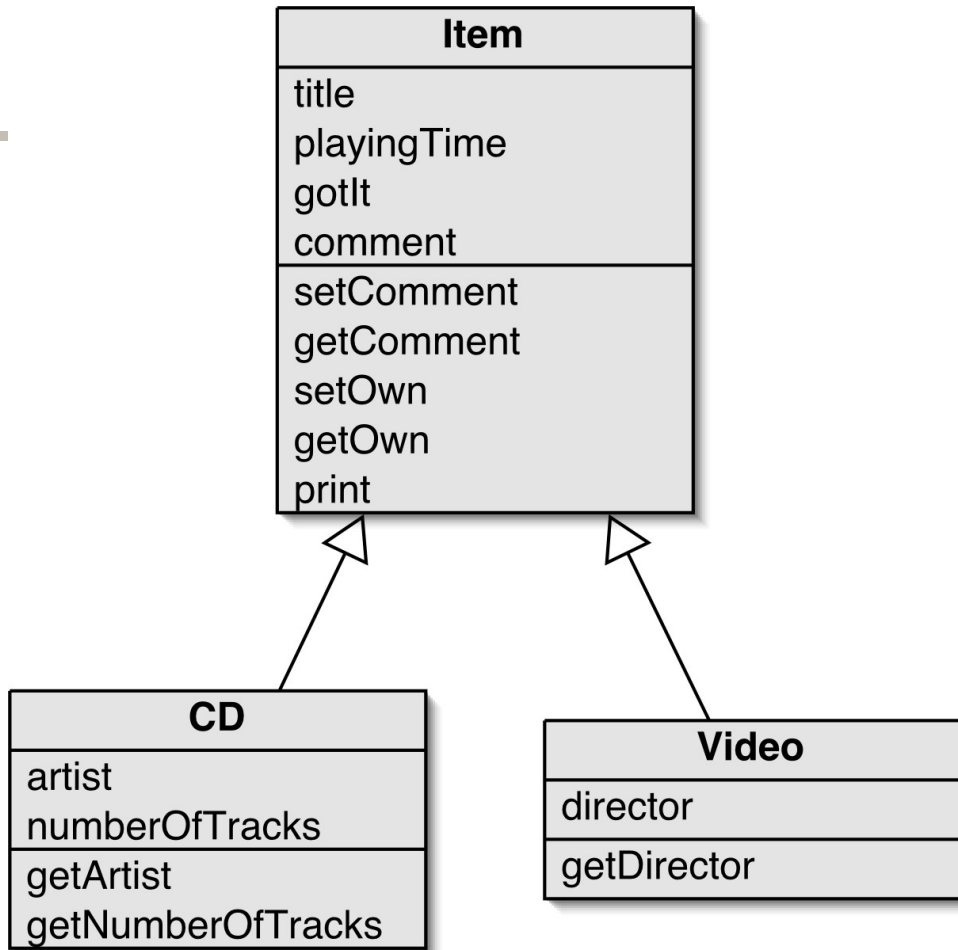
---

## Code re-use through hierarchies

```
public class Person{
    String name;
    int id;
    public String getName(){
        return name;
    }
}
```

```
Faculty f = new Faculty();
String f_name = f.getName();
Student s = new Student();
String s_name = s.getName();
```

```
public class Faculty extends Person{
    ArrayList<Class> lectures;
}
public class Student extends Person{
    ArrayList<Class> classes;
}
public class Staff extends Person{
    Staff boss;
}
```



# Inheritance – What is it?

---

Definition: The process by which one class acquires the properties and methods of another.

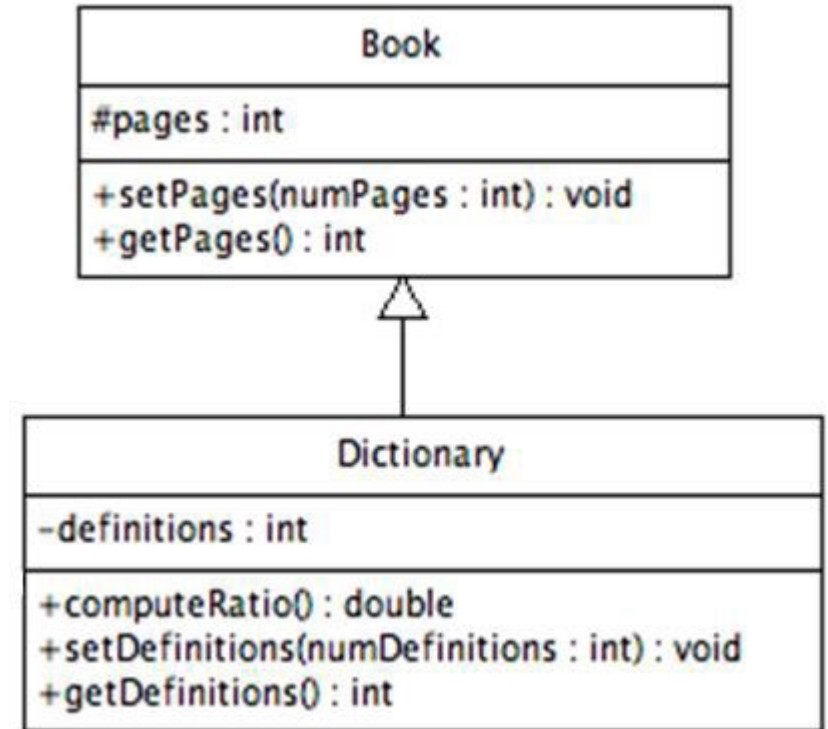
```
public class <class_1> extends <class_2> {  
}
```

## Terminology

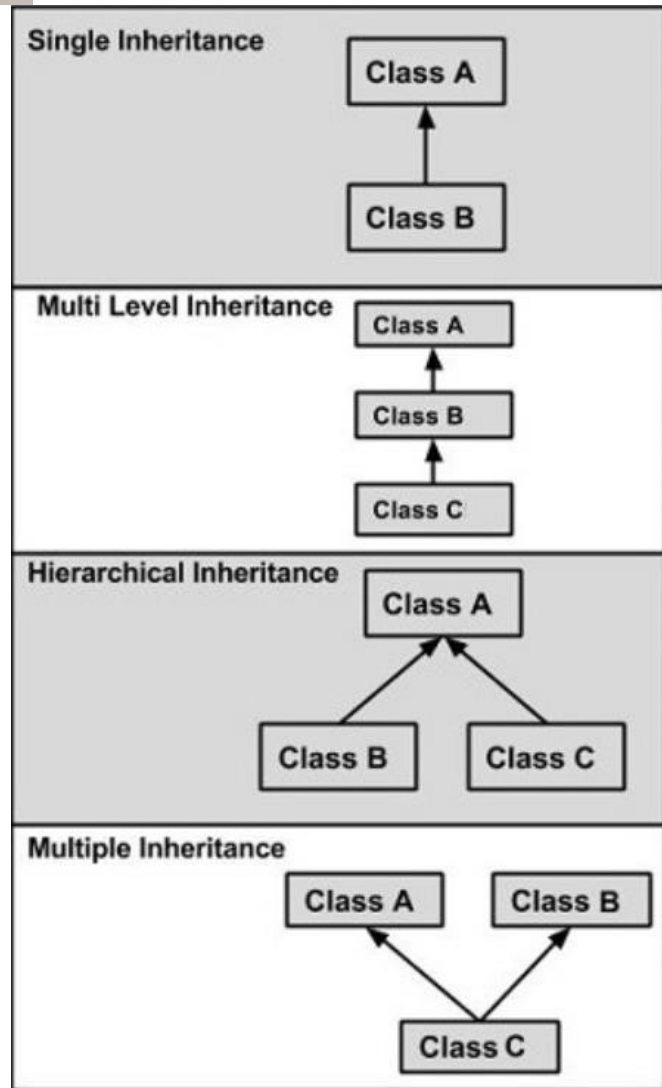
- We call the derived class <class\_1> the **sub** or **child** class
- We call the base class <class\_2> the **super** or **parent** class
- **IS-A** relationship
  - **sub** class is-a **super** class, **child** is-a **parent**
- Boolean operator **instanceof** - > {True, False}
  - student\_instance **instanceof** Person ---> **True**
  - person\_instance **instanceof** Student ---> **False**
  - student\_instance **instanceof** Student ---> **True**
  - person\_instance **instanceof** Person ---> **True**

# Inheritance – What is it?

- inheritance creates an **is-a** relationship
  - the child is a more specific version of the parent
- you can view these as a **family** of classes
  - some variables / methods defined only once and yet apply to the whole family
- **Software reuse** is a major benefit of inheritance



# Types of Inheritance



```
public class A{...}  
public class B extends A{...}
```

```
public class A{...}  
public class B extends A{...}  
public class C extends B{...}
```

```
public class A{...}  
public class B extends A{...}  
public class C extends A{...}
```

```
public class A{...}  
public class B{...}  
public class C extends A, B{...}
```

Java does not have multiple inheritance

# Surprise you've been using it all along

---

Every java class is descended from the super class Object

It's been hidden from you all along

```
public class Person {...} -> public class Person extends Object {...}
```

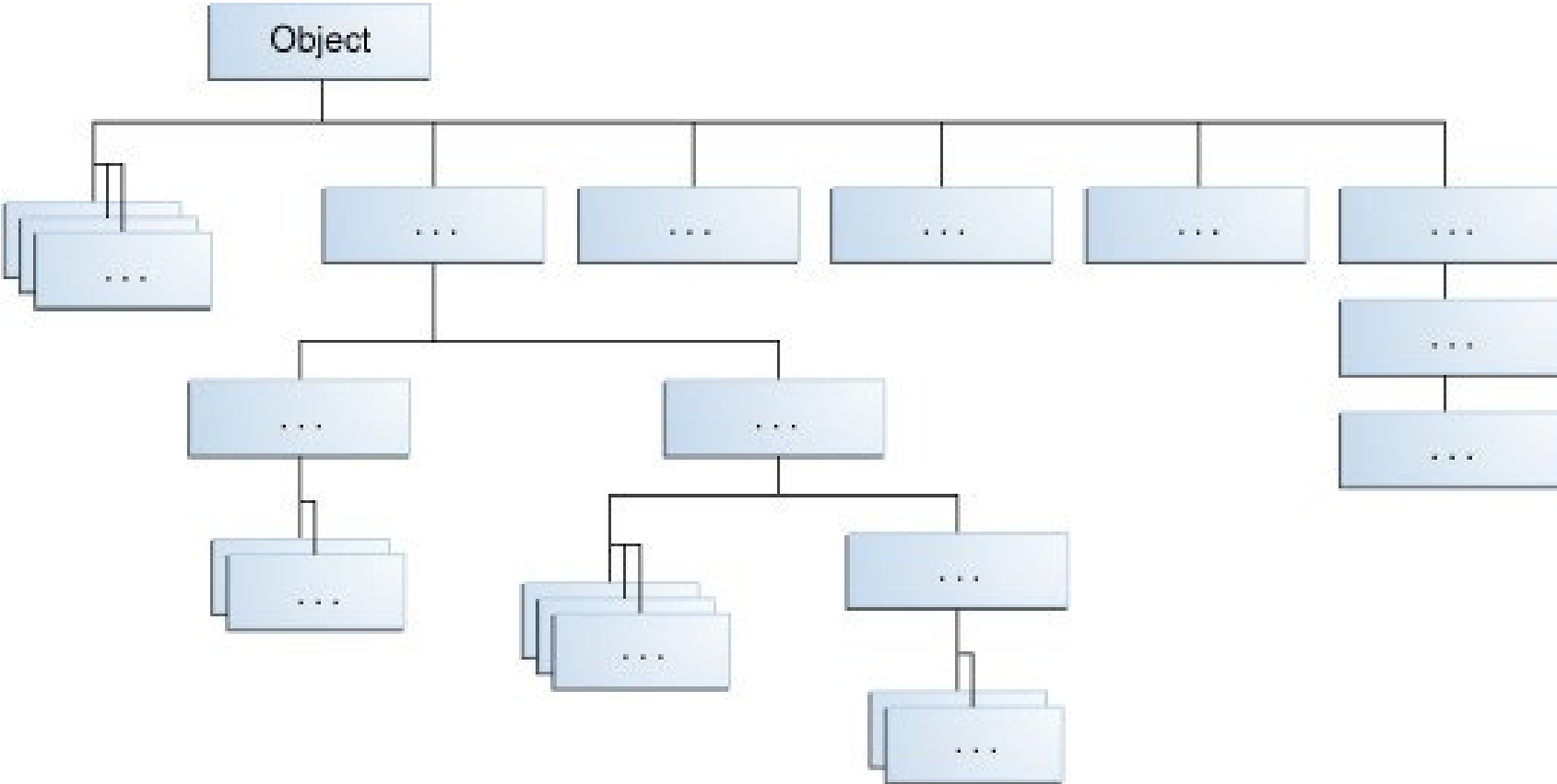
<https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>

<code>boolean</code>	<code>equals(Object obj)</code> Indicates whether some other object is "equal to" this one.
<code>Class&lt;?&gt;</code>	<code>getClass()</code> Returns the runtime class of this Object.
<code>int</code>	<code>hashCode()</code> Returns a hash code value for the object.
<code>String</code>	<code>toString()</code> Returns a string representation of the object.



# Hierarchy

---



# The protected Modifier

---

- private variables/methods can **not** be referenced by name in a child class
- public variables/methods can be – but public variables violate the principle of encapsulation
- **protected** visibility often used for inheritance
  - allows a child class to reference a variable or method directly in the child class

# private/protected/public

Private -> only class can see it

Protected -> only class and sub-classes can see it

Public -> everyone can see it

```
public class Person{
    private int private_id;
    protected int protected_id;
    public int public_id;
    int undeclared_id;
}
public class Student extends Person{
    //I can get at protected_id, public_id, undeclared_id
}
public class Class{
    //I can get at public_id
}
```

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

# The protected Modifier

---

- Are there any disadvantages?
  - The super and sub classes are more tightly coupled (changing super may involve rewriting sub)
  - also visible to any class in the same package as the parent class (may be a problem)
- Recommendations
  - leave instance variables private
  - Access them through inherited methods
  - Protected helper methods may be useful

```
public class Person {
    protected String name;
    private int id;
    public int getId() {
        return id;
    }
}
public class Faculty extends Person {
    private ArrayList<Session> lectures;
    public String toString() {
        return String.format("%s %s %s", name, getId(), lectures);
    }
}
public class Staff extends Person {
    private Staff boss;
    public String toString() {
        return String.format("%s %s %s", name, getId(), boss);
    }
}
public class Student extends Person {
    private ArrayList<Session> classes;
    public String toString() {
        return String.format("%s %s %s", name, getId(), classes);
    }
}
```

# The super Reference

---

- The `super` reference can be used to reference variables and methods defined in the parent's class
- Constructors are not inherited; each class should have its own
- should use `super` to invoke the parent's constructor to set up the “parent's part” of the object
  - must be the **first** line of a child's constructor

# Inheritance and Constructors

---

- The `super` reference can be used to reference variables and methods defined in the parent's class
- Constructors are not inherited; each class should have its own
- should use `super` to invoke the parent's constructor to set up the “parent's part” of the object
  - must be the **first** line of a child's constructor
- Saves you time re-implementing large constructors with shared code

# Inheritance and Constructors

---

```
public class Person {  
  
    protected String name;  
    protected int id;  
  
    public Person(String name, int id) {  
        this.id = id;  
        this.name = name;  
    }  
}  
  
public class Student extends Person {  
  
    private ArrayList<Session> classes;  
  
    public Student(String name, int id, ArrayList<Session> classes) {  
        super(name, id);  
        this.classes = classes;  
    }  
}
```



# Onward to ... Overriding

---

Jonathan Hudson  
[jwhudson@ucalgary.ca](mailto:jwhudson@ucalgary.ca)  
<https://pages.cpsc.ucalgary.ca/~jwhudson/>



UNIVERSITY OF  
CALGARY