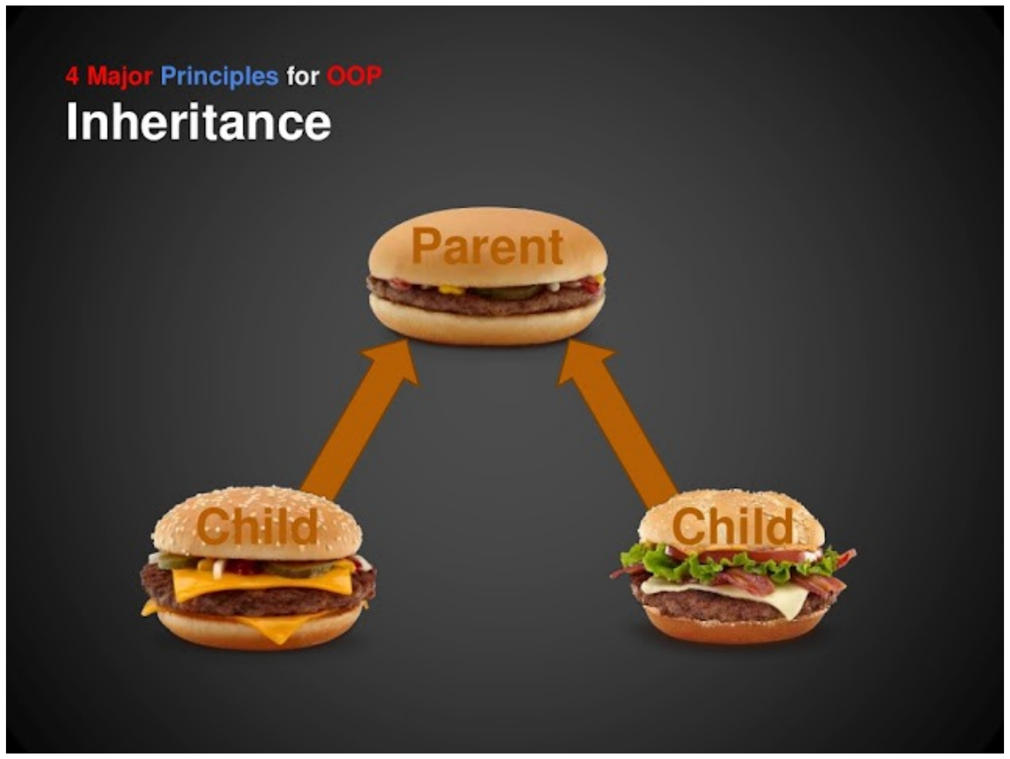
Inheritance – Part 2

CS 171: Intro to Computer Science II



Inheritance Code Examples: Agenda

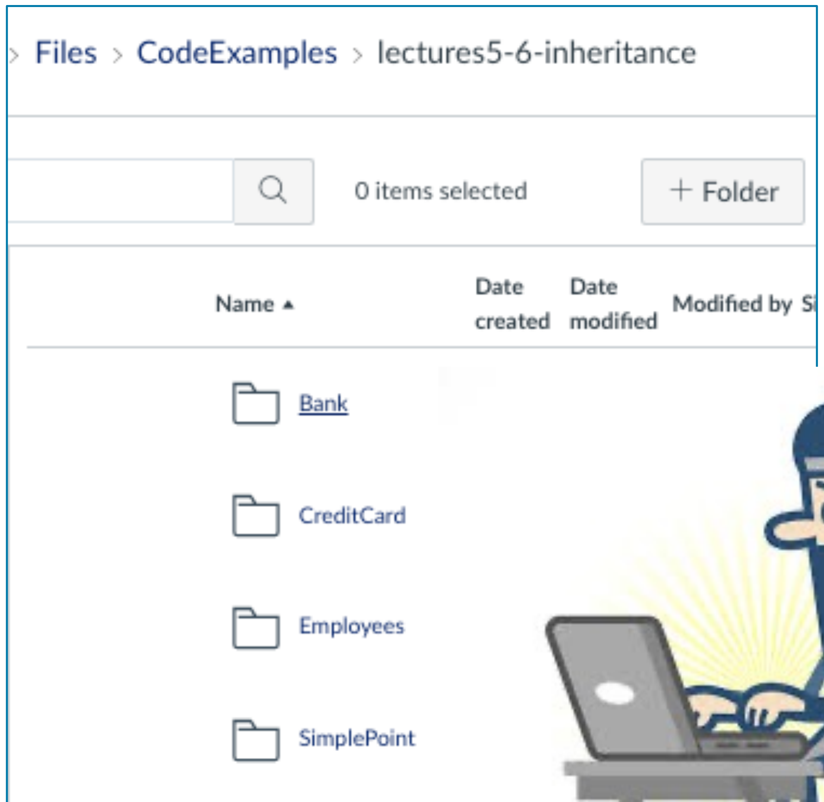
[Let's continue this!] 1. Start by simply extending Point to Point3D (a point that has 3 coordinates: x, y, z)

More exercises:

2. Credit card application with different type of cards

3. Company application with different types of Employees (salaried, hourly)

4. Bank application with different types of accounts (checking, savings)


Files › CodeExamples › lectures5-6-inheritance

0 items selected

Bank

CreditCard

Employees

SimplePoint

Employee example

[Source code available on Canvas]

```java
/** a generic employee class */
public class Employee {
    private String name; // name of the employee
    public Employee(String n) { name = n; }
    public Employee() { name = "Unknown"; }
    public String getName() { return name; }
    public String toString() { return name; }
    public double earnings() { return 0; }
}
```

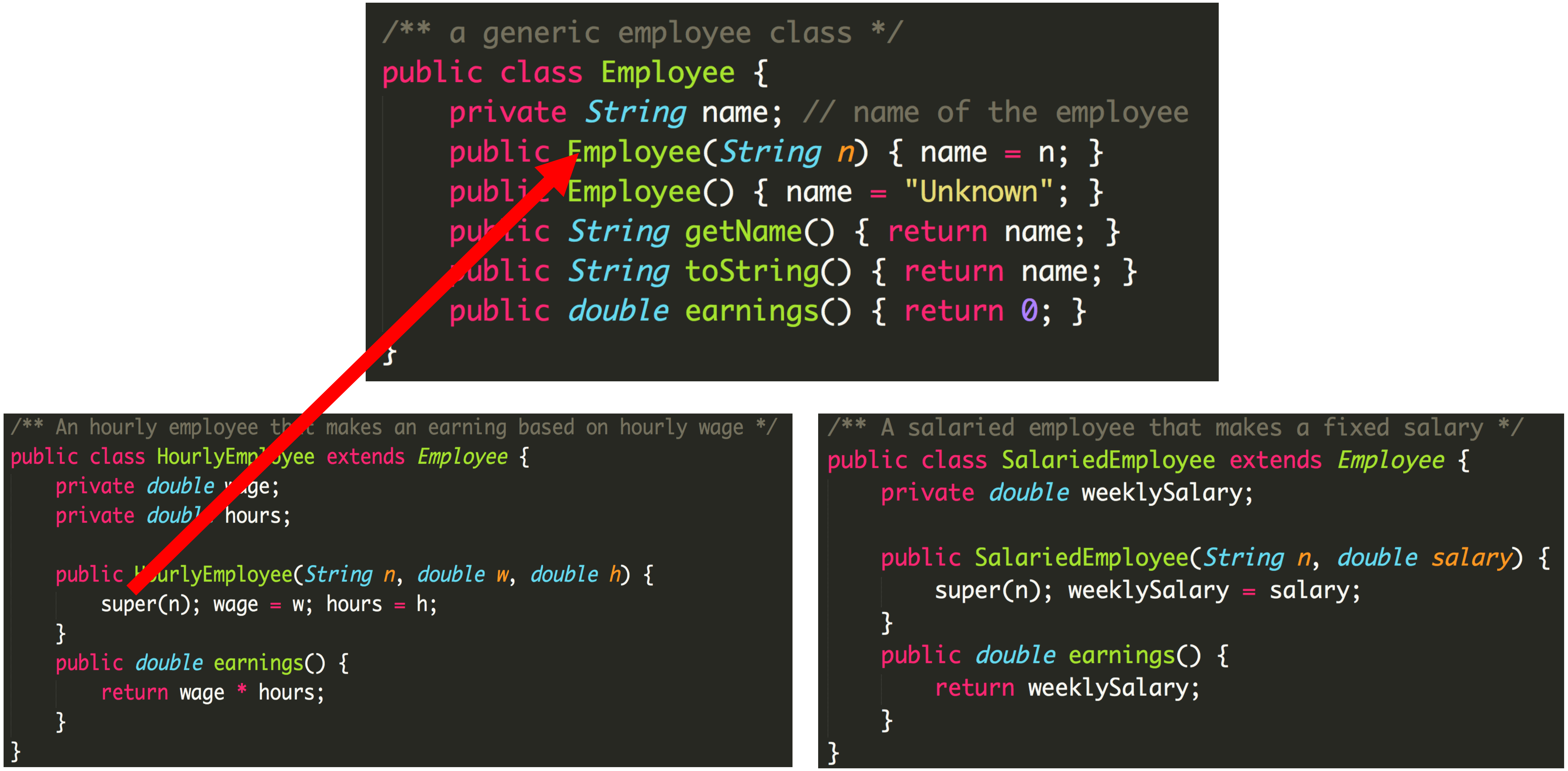
```java
/** An hourly employee that makes an earning based on hourly wage */
public class HourlyEmployee extends Employee {
    private double wage;
    private double hours;

    public HourlyEmployee(String n, double w, double h) {
        super(n); wage = w; hours = h;
    }
    public double earnings() {
        return wage * hours;
    }
}
```

```java
/** A salaried employee that makes a fixed salary */
public class SalariedEmployee extends Employee {
    private double weeklySalary;

    public SalariedEmployee(String n, double salary) {
        super(n); weeklySalary = salary;
    }
    public double earnings() {
        return weeklySalary;
    }
}
```

Employee example
[Source code available on Canvas]



```java
/** a generic employee class */
public class Employee {
    private String name; // name of the employee
    public Employee(String n) { name = n; }
    public Employee() { name = "Unknown"; }
    public String getName() { return name; }
    public String toString() { return name; }
    public double earnings() { return 0; }
}
```

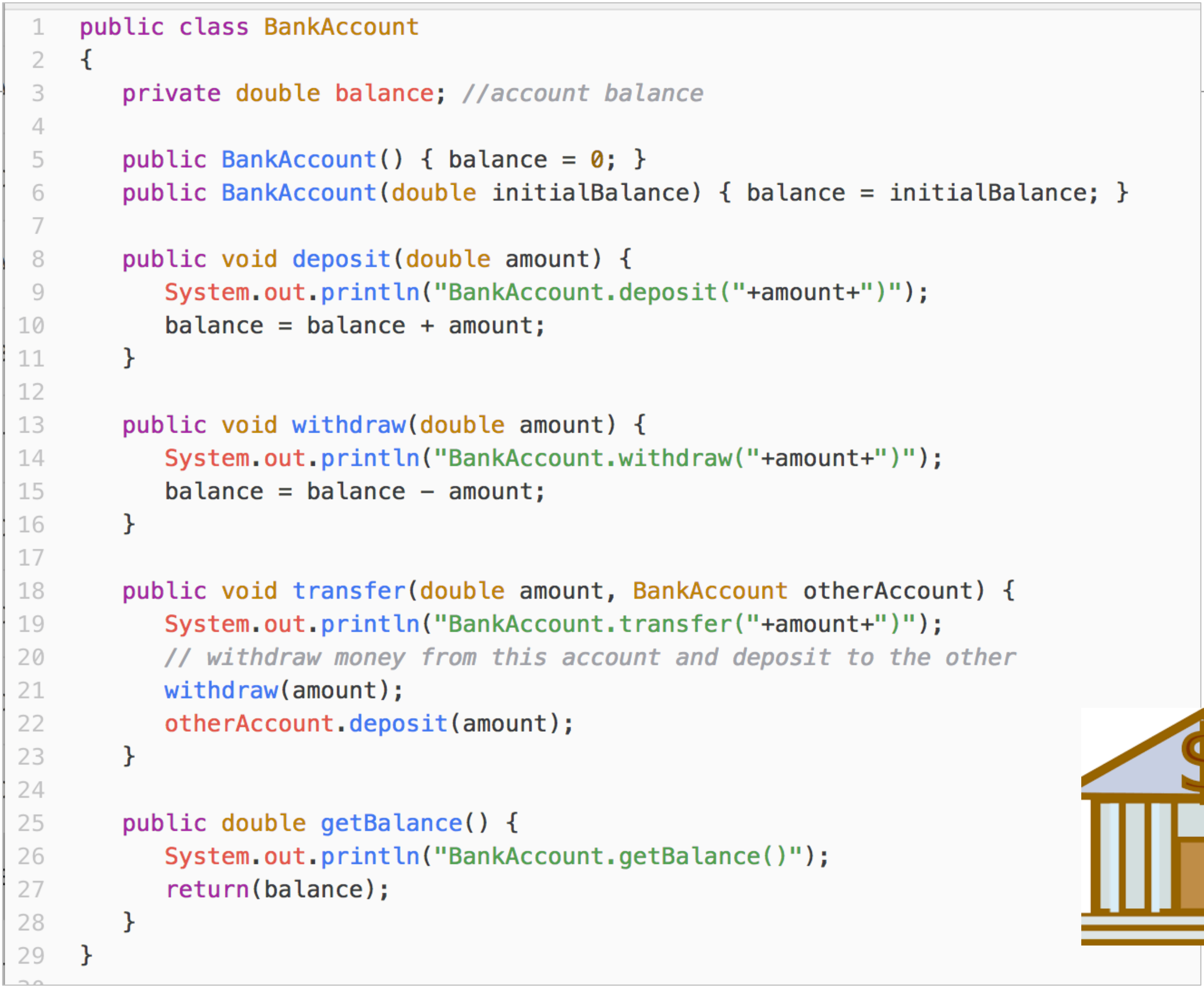
```java
/** An hourly employee that makes an earning based on hourly wage */
public class HourlyEmployee extends Employee {
    private double wage;
    private double hours;

    public HourlyEmployee(String n, double w, double h) {
        super(n); wage = w; hours = h;
    }
    public double earnings() {
        return wage * hours;
    }
}
```

```java
/** A salaried employee that makes a fixed salary */
public class SalariedEmployee extends Employee {
    private double weeklySalary;

    public SalariedEmployee(String n, double salary) {
        super(n); weeklySalary = salary;
    }
    public double earnings() {
        return weeklySalary;
    }
}
```

Bank Example – Super Class



```java
public class BankAccount
{
   private double balance; //account balance

   public BankAccount() { balance = 0; }
   public BankAccount(double initialBalance) { balance = initialBalance; }

   public void deposit(double amount) {
      System.out.println("BankAccount.deposit("+amount+")");
      balance = balance + amount;
   }

   public void withdraw(double amount) {
      System.out.println("BankAccount.withdraw("+amount+")");
      balance = balance - amount;
   }

   public void transfer(double amount, BankAccount otherAccount) {
      System.out.println("BankAccount.transfer("+amount+")");
      // withdraw money from this account and deposit to the other
      withdraw(amount);
      otherAccount.deposit(amount);
   }

   public double getBalance() {
      System.out.println("BankAccount.getBalance()");
      return(balance);
   }
}
```

CheckingAccount

- Customer has a limited number of FREE transactions (i.e. deposits or withdraws) = 2
- After reaching that limit, customer pays an additional fee with every transaction

Bank Example - Subclass

```java
public class CheckingAccount extends BankAccount
{
   /* Static variables that are consistent across all checking accounts */
    private static final int FREE_TRANSACTIONS = 2;
    private static final double TRANSACTION_FEE_MULTIPLIER = 2.0;

    private int transactionCount;

    public CheckingAccount(double initialBalance) {
      super(initialBalance);
      transactionCount = 0;
   }

   @Override
   public void deposit(double amount) {
      //Override deposit to keep track of transactions
      transactionCount++;
      super.deposit(amount);
   }

   @Override
   public void withdraw(double amount) {
      //Override withdraw to keep track of transactions
       transactionCount++;
      super.withdraw(amount);
   }

   public void deductFees()
   {
     if (transactionCount > FREE_TRANSACTIONS) {
        double fees = TRANSACTION_FEE_MULTIPLIER * (transactionCount - FREE_TRANSACTIONS);
        withdraw(fees);
      }
      transactionCount = 0;
    }
}
```

Inherits “balance” but creates new data members

Subclass constructor calls superclass constructor, then initializes its own private var.

compiler annotation

Overriding deposit() so that we add an extra calculation before calling the original deposit

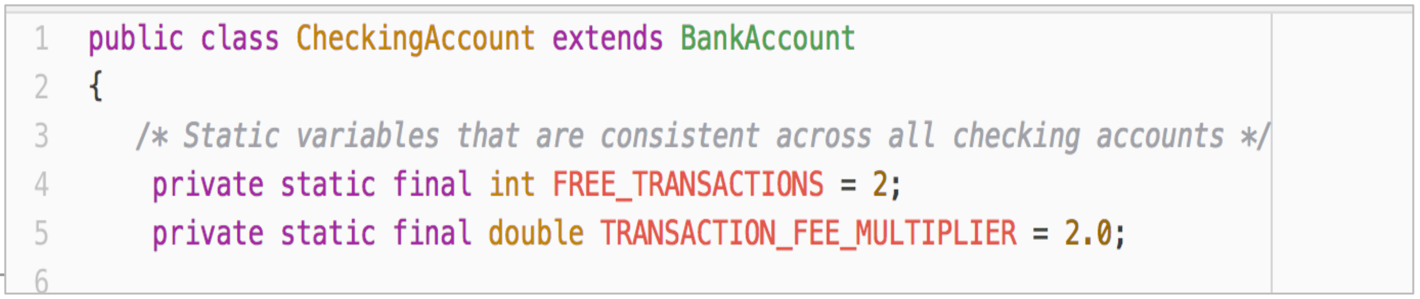
I can also call super.withdraw(fees) here

Checking Account Class

- Instance fields
  - `balance` — inherited from BankAccount but private
  - `transactionCount` —new to Checking Account
- Methods
  - `getBalance()` — inherited from BankAccount
  - `deposit(double amount), withdraw(double amount)` — override and update transaction count

Now what's the deal with this `static` *keyword, and when do we use it?*

`static` variable



```java
public class CheckingAccount extends BankAccount
{
   /* Static variables that are consistent across all checking accounts */
    private static final int FREE_TRANSACTIONS = 2;
    private static final double TRANSACTION_FEE_MULTIPLIER = 2.0;

```

- Used for common property of all objects — not unique for each object

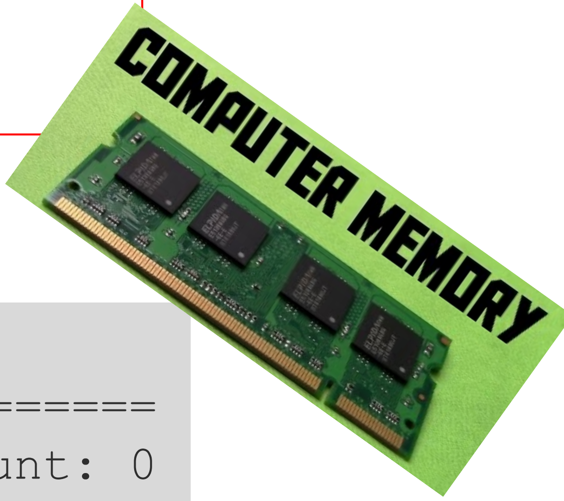
- Static variable gets memory once at the time of class loading — saves memory!

```
CheckingAccount c1 = new CheckingAccount(100);
CheckingAccount c2 = new CheckingAccount(200);
CheckingAccount c3 = new CheckingAccount(300);
```

```
FREE_TRANSACTIONS:  2       STATIC VARIABLES
TRANSACTION_FEE_MULTIPLIER: 2.0
```

```
         c1                         c2                         c3
====================       ====================       ====================
transactionCount: 0        transactionCount: 0        transactionCount: 0

--------------------       --------------------       --------------------
    getBalance()               getBalance()               getBalance()
<more methods here…>       <more methods here…>       <more methods here…>
```



`static` variable

- In this previous Bank example, the static variables were also `final`
- A static variable that is not final can be modified!
  - If any object modifies the value of that static variable, other objects of this class will see the updated value!

(See the BankAccount code example uploaded on Canvas)

`static` method

- Static method belongs to class rather than object of class
- Can be invoked without creating an instance of a class
- Can access static data member and change value of it

When do I declare a method as static, vs leave it as an instance method (non-static)?

- Let’s think of this question in the context of the bank’s CheckingAccount class ….

`static` block

- Block can be used to initialize the static data member
  - Executes before main method
  - Syntax:
    ```
    static { // statements };
    ```

CreditCard Class

[Complete Class and Tester Code: Textbook Pages 42-43]

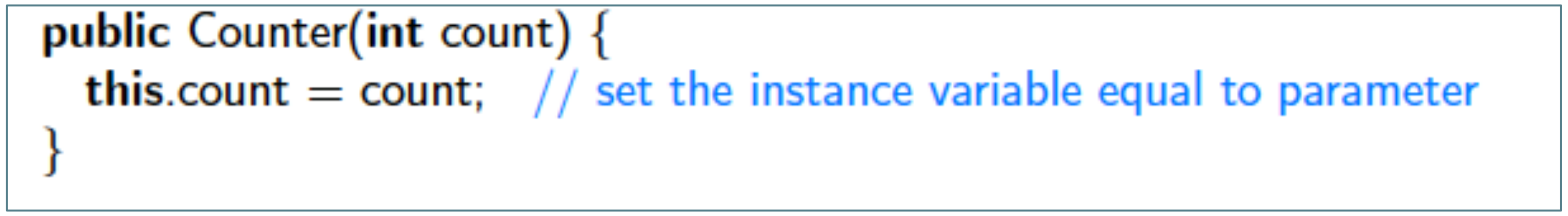
```
public class CreditCard {
  // Instance variables:
  private String customer;       // name of the customer (e.g., "John Bowman")
  private String bank;           // name of the bank (e.g., "California Savings")
  private String account;        // account identifier (e.g., "5391 0375 9387 5309")
  private int limit;             // credit limit (measured in dollars)
  protected double balance;      // current balance (measured in dollars)
  // Constructors:
  public CreditCard(String cust, String bk, String acnt, int lim, double initialBal) {
    customer = cust;
    bank = bk;
    account = acnt;
    limit = lim;
    balance = initialBal;
  }
  public CreditCard(String cust, String bk, String acnt, int lim) {
    this(cust, bk, acnt, lim, 0.0);                 // use a balance of zero as default
  }
```

More Keywords…

`this` Keyword

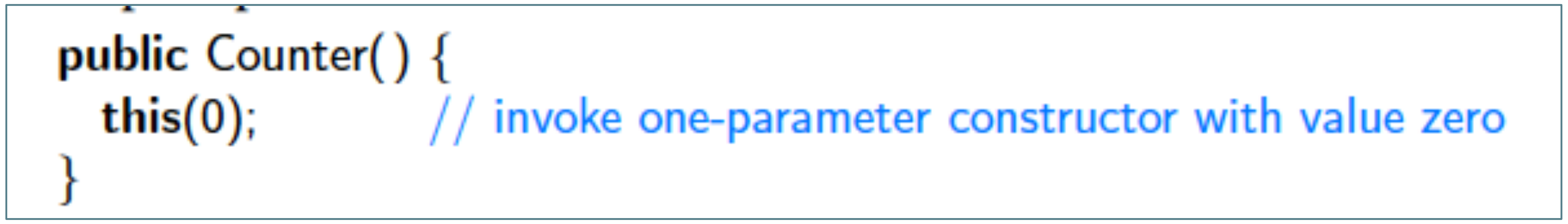
- Reference to the <u>current</u> object
- Usage:

1. Differentiate between an instance variable and local variable with the same name



```
public Counter(int count) {
  this.count = count;   // set the instance variable equal to parameter
}
```

2. To allow one constructor to invoke another constructor (avoids code reusage in some cases)



```
public Counter() {
  this(0);          // invoke one-parameter constructor with value zero
}
```

`final` Keyword

- `final` variables
  - Initialized as a part of declaration but never can get a new value (i.e. a constant)
  - A final field of a class will almost always be static (wasteful to have every instance store an identical value!)
- `final` methods
  - Cannot be overridden by a subclass
- `final` classes
  - Cannot be subclassed

Inheritance Hierarchies

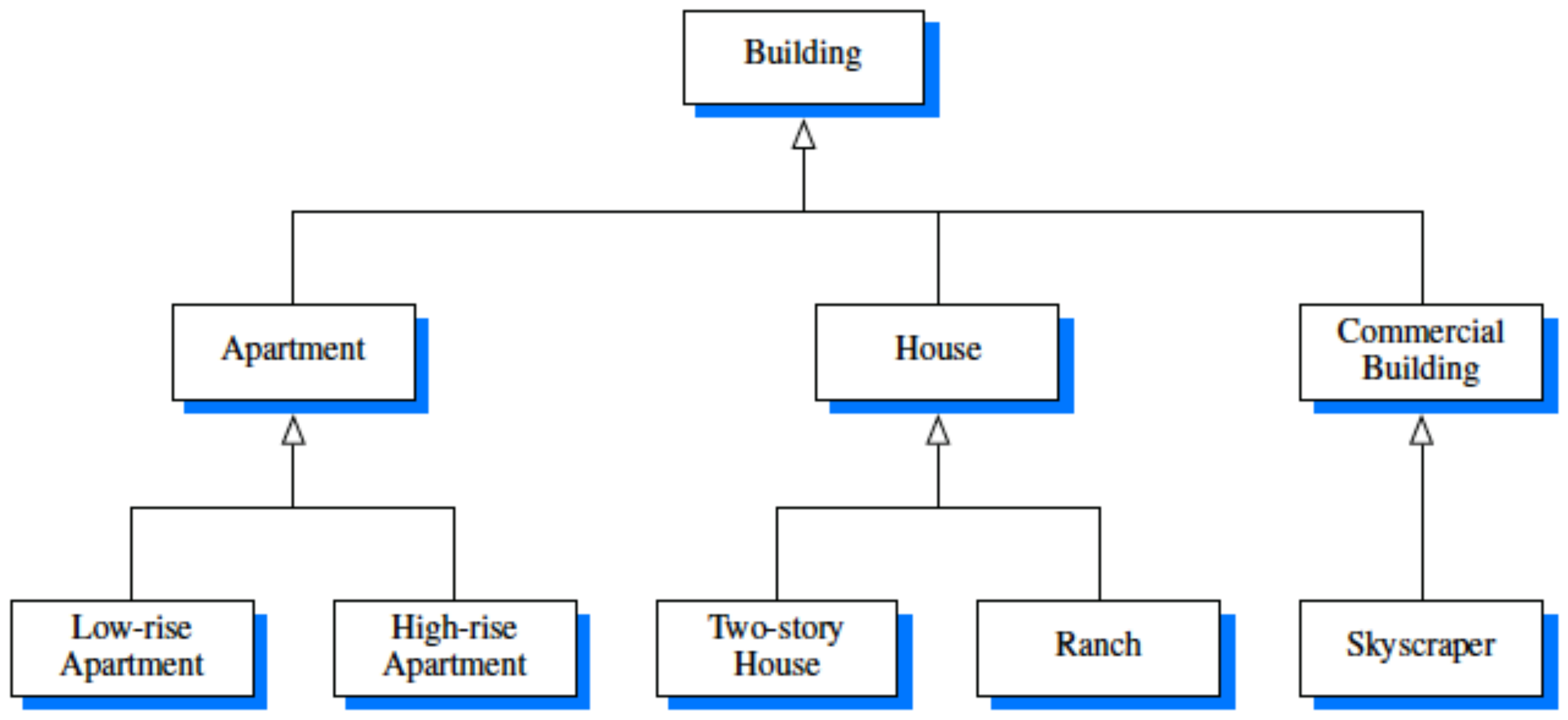


Figure 2.3: An example of an “is a” hierarchy involving architectural buildings.

- An *apartment* is a *building*.
- A *ranch* is a *house*.
- A *ranch* is a *building*.
- A *building* is NOT (always) a *ranch*.
- A *house* is NOT a *skyscraper*.

Question

`private int x` can be accessed within:

✓ A. The same class as x

B. The same class as x and any subclasses

C. The same class as x, any subclasses, and other files in the same folder

D. Any Java class anywhere in the world