OOJ Lecture 3 Inheritance

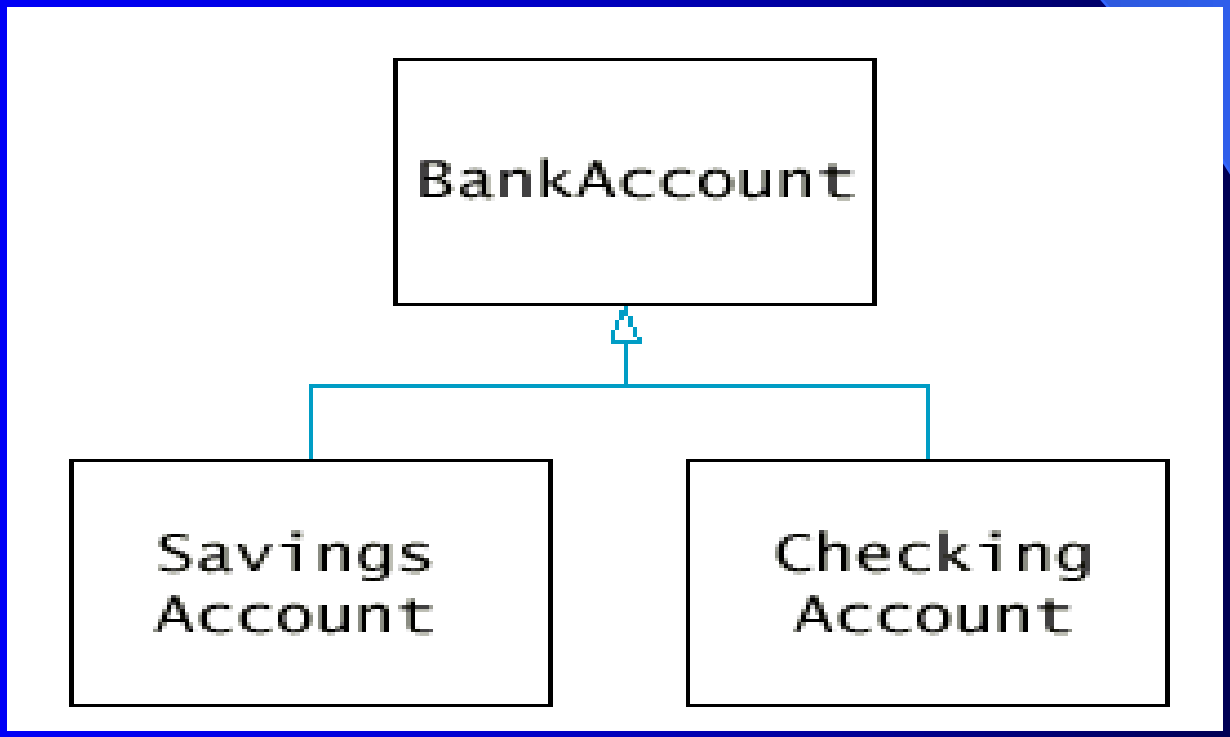
- Reading: Savitch, Chapter 7
- Reference: Big Java, Horstman, Chapter 11

Objectives

- To further explore inheritance and the overriding of superclass methods
- A Case study
- To learn about private field access control

Bank Account Hierarchy

- **We will study a simple bank account hierarchy**



Example: Superclass BankAccount

```
//Reference: Horstmann, Chapter 11
public class BankAccount
{   public BankAccount(double initialBalance)
    {
       balance = initialBalance;
    }
    public void deposit(double amount)
    {
       balance = balance + amount;
    }
    public void withdraw(double amount)
    {
       balance = balance - amount;
    }
```

Example: Superclass BankAccount

```
    public double getBalance()
    {
       return balance;
    }
    public void transfer(BankAccount b1, double
    amount)
    {
       withdraw(amount);
       b1.deposit(amount);
    }

    private double balance;
}
```

Adding a Subclass Method

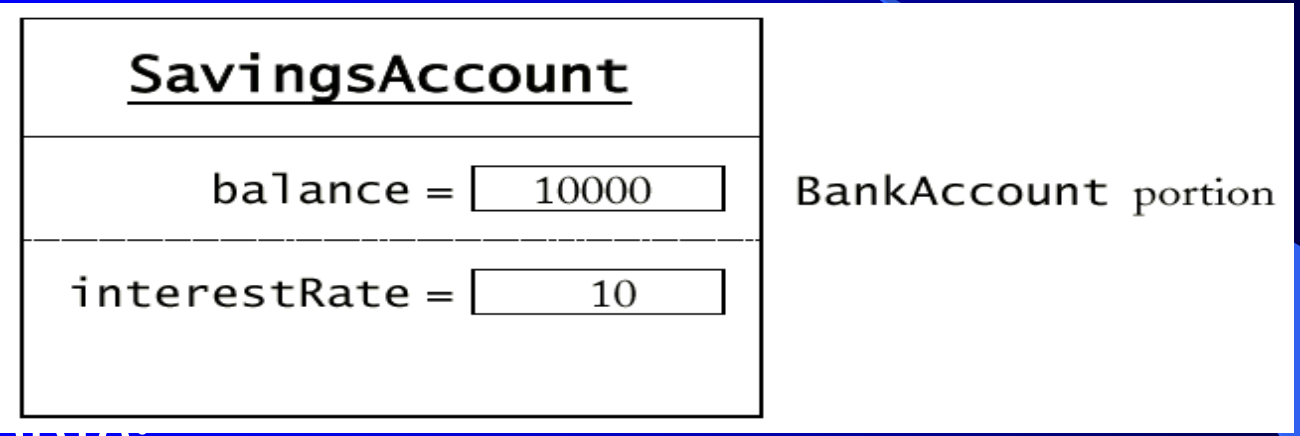
```
//Reference: Horstmann, Chapter 11
public class SavingsAccount extends BankAccount
{
    public SavingsAccount(double rate)
    //constructor
    {   interestRate = rate; }
    public void addInterest()
    {
     double interest = getBalance()*interestRate/100;
     deposit(interest);
    }

    private double interestRate;
}
```

- Subclasses should extend superclasses by adding or redefining methods, and adding instance fields

Class Exercise

- If you entered the code from slides 4,5 and 6, then created a driver program, the SavingsAccount constructor would not compile.
- Why?
- There are 2 ways to fix this problem, what are they?
- Hint: re-read slide 11 from lecture 2

Layout of a Subclass Object



- Example:

```
SavingsAccount myAccount = new SavingsAccount(10);
myAccount.addInterest();
```

→ `double interest = getBalance()*interestRate/100;`
`deposit(interest);`

- The $1000 will be deposited to myAccount.

CheckingAccount Class

- Implement the subclass CheckingAccount
  - First three transactions are free
  - Charge $2 for every additional transaction
  - Must override `deposit`, `withdraw` to increment transaction count
  - A `deductFees` method deducts accumulated fees, resets transaction count

CheckingAccount Class

Three methods must be overridden or defined:

```
//Reference: Horstmann, Chapter 11
public class CheckingAccount extends BankAccount
 {
        public CheckingAccount(int initialBalance){..}
    //constructor
        public void deposit(double amount) {..}   //override
        public void withdraw(double amount) {..} //override
        public void deductFees() {..}   //new
      …
        private int transactionCount;
}
```

A subclass has no direct access to private fields of its superclass

Calling Superclass Construction

```
//Reference: Horstmann, Chapter 11
public class CheckingAccount extends BankAccount
{  public CheckingAccount(double initialBalance)
   {
         // construct superclass
      super(initialBalance);
         // initialize transaction count
      transactionCount =0;
   }
   ...
}
```

Pass parameters to superclass constructor as the part of subclass constructor

- Remember to use super as much as possible!
- Do not repeat code already available in superclass!

Inherited Fields are Private

Using examples from last lecture again:

```
// Reference: Horstmann, Chapter 11
public class CheckingAccount extends BankAccount
 {
        public CheckingAccount(int initialBalance){..} //constructor
        public void deposit(double amount) {..}   //override
        public void withdraw(double amount) {..} //override
        public void deductFees() {..}   //new
    …
        private int transactionCount;
}
```

A subclass has no direct access to private fields of its superclass, such as *balance* from *BankAccount*

Inherited Fields are Private

Remember the deposit method in class BankAccount:

```
public class BankAccount
{
  public BankAccount(double initialBalance)
  { …}
  public void deposit(double amount)
  {    balance = balance + amount;  } //change balance
  public void withdraw(double amount)
  { …  }
  public double getBalance()
  {…  }
   private double balance;
}
```

subclass can only call methods deposit or withdraw to change private variable *balance*

Inherited Fields are Private

- Consider `deposit` method of `CheckingAccount`, it needs to count the number of transactions and deposit the money.
- If we override with the following

```
public void deposit(double amount)
{
   transactionCount++;
   balance = balance + amount // Wrong!!
   ...
}
```

- Can't just add `amount` to `balance` (not accessible)
- `balance` is a *private* field of the superclass

Inherited Fields are Private

Subclasses must use a public method of superclass to access private variables:

- use the method deposit from superclass to change balance
- Since `deposit` method is overridden in `CheckingAccount`
- you can't just call
  `deposit(amount)`
  in `deposit` method of `CheckingAccount`

```
public void deposit(double amount)
{   transactionCount++;
    deposit(amount); //Wrong
}
```

Invoking a Superclass Method
to access private variables of superclass

- That is the same as

```
this.deposit(amount)
```

- It will cause infinite recursion since it is calling itself
- Should invoke *superclass method*

```
super.deposit(amount)
```

- This calls the `deposit` method of `BankAccount` class, not deposit method in derived class

`private` & `public` Instance Variables and Methods

- `private` instance variables from the superclass are not available by name in subclasses
  - "Information Hiding" says they should not be seen
  - use accessor methods to change them, e.g. `deposit(.)` for a `BankAccount` object to change the `balance` of the account
- `private` methods are **not** inherited!
  - use `public` to allow methods to be inherited
  - only helper methods should be declared `private`

Class Exercise

- Write a subclass *HourlyEmp* for the following Employee class. The hourly employees are paid by hours with an hourly rate.
- How do you get the salary of hourly employees?

```
public class Employee
{  private double salary;
   private String name;
   public Employee (String aName, double aSalary)
   {          ...                    }
   public double getSalary()
   {     return salary;               }
   public void setSalary(double newSalary)
   {    salary = newSalary;           }
   public void writeOutput()
   {    System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);}
```

Class Exercise

```
public class HourlyEmp extends Employee
{
  private int hours;
  public HourlyEmp(String aName, int hours)
  { // fill in the constructor  }
  public double calculateSalary()
  {
     // NOTE: Salary is a private variable
     // of Employee.
```

You may use setSalary(newSalary) of Employee to assign the salary of hourly employees.

```
  }
}
```