CHAPTER 10
OBJECT INHERITANCE

Introduction to Computer Science Using Ruby

Inheritance

- Classes can be defined so as to have **relationships** with other classes
- The most basic of these relationships is called **inheritance**
  - No need to redefine similar parts of classes
  - A class can **inherit properties** from another class
  - Inheritance can represent the relationship between a generic ball, a baseball, a tennis ball, and a ping pong ball: they are all spherical objects

Inheritance

Account class:

- We expand our view of accounts to create both a checking and a savings account
- Analyze what they have in common

Attributes Shared:

- Have a balance
- Can withdraw money
- Can deposit money

Inheritance

Parent Class or Superclass:

- Defines **attributes** that both types of accounts can use
- Defines the **similarities** in the relationship
- Eliminates the need to duplicate common data and methods

Child Class or Subclass:

- Defines the **differences**
  - i.e., checking and savings accounts
- Main differences:
  - Cannot withdraw beyond the minimum balance from a savings account
  - Savings account generates interest

Inheritance

- The checking and savings account classes will define the differences
  - These are the **child class** or **subclass**
- The main differences are:
  - You cannot withdraw beyond the minimum balance from a savings account
  - A savings account generates interest

Example 10.1: Savings Account Version #1

```
 1 require_relative '../chapter_09/account_5.rb'
 2
 3 class SavingsAccount < Account
 4   def initialize(balance, name, phone_number,
interest, minimum)
 5     super(balance, name, phone_number)
 6     @interest = interest
 7     @minimum = minimum
 8   end
 9
10   def accumulate_interest
11     @balance += @balance * @interest
12   end
13 end
```



```
class Account
    def initialize(balance, name, phone_number)

        @balance = balance
        @name = name
        @phone_number = phone_number
    end

    def deposit(amount)
        @balance += amount
    end

    def withdraw(amount)
        @balance -= amount
    end
```



```
    def display
        puts "Name: " + @name
        puts "Phone number: " + @phone_number.to_s
        puts "Balance: " + @balance.to_s
    end

    def transfer(amount, target_account)
    @balance -= amount
    target_account.deposit(amount)
    end

    def status
        return @balance
    end

end
```

```
 1 require_relative '../chapter_09/
 2
 3 class SavingsAccount < Account
 4   def initialize(balance, name, phone_number,
interest, minimum)
 5     super(balance, name, phone_number)
 6     @interest = interest
 7     @minimum = minimum
 8   end
 9
10   def accumulate_interest
11     @balance += @balance * @interest
12   end
13 end
```

< defines inheritance in Ruby

Inheritance

- The SavingsAccount class can do more than the method it defined
  - It **inherits** all of the super class' variables and methods

Table 10.1: SavingsAccount Inherited Summary

Data	Methods
@balance	withdraw(amount)
@name	deposit(amount)
@phone_number	transfer(amount,targetAmount)
	display

Inheritance: Polymorphism

- Because SavingsAccount is subclass of Account, it can use the **transfer method** to send funds to an Account object
- Does have its limits
  - Cannot use the subclasses properties on a superclass
  - Subclass has features the superclass does not
  - Cannot use accumulate_interest() method on an Account object

Basic Methods Overriding

- It is sometimes convenient to alter methods that already exist in a superclass
- The SavingsAccount class needs to make sure the balance does not go below the minimum
  - To achieve this, the SavingsAccount class will need to **override** the withdraw method
  - Needs to **define** its own withdraw functionality

Example 10.2: SavingsAccount Version #2

```
1 require_relative '../chapter_09/account_5.rb'
2
3 class SavingsAccount < Account
4    def initialize(balance, name, phone_number,
  interest, minimum)
5       super(balance, name, phone_number)
6       @interest = interest
7       @minimum = minimum
8    end
9
10   def accumulate_interest
11      @balance += @balance * @interest
12   end
13
```

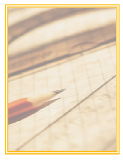

Accessing the Superclass

- In many cases, the overriding methods are similar to the methods they override
- Instead of repeating code, we can call the superclass inside an **overridden method**
  - Simply insert the word **super** with all the parameters that are needed

Example 10.3: SavingsAccount Version #3

```
1 require_relative '../chapter_09/account_5.rb'
2
3 class SavingsAccount < Account
4   def initialize(balance, name,
    phone_number, interest, minimum)
5      super(balance, name, phone_number)
6      @interest = interest
7      @minimum = minimum
8   end
9
10  def accumulate_interest
11     @balance += @balance * @interest
12  end
13
```


Summary



- **Inheritance:** classes can be created from other classes and use the resources of the parent class
- The parent class or the superclass, **defines the relationship** with the child class or subclass
- Subclasses inherit **both data and methods** from their parent class
- In some cases, methods used by the child class need to be **overridden**