**CHAPTER 9: INHERITANCE**

**Chapter 9-A: Inheritance**

**There are four major principles of object-oriented programming:**

* encapsulation (in chapter 6)
* inheritance
* polymorphism
* abstraction

In this chapter, we will focus solely on ***inheritance***.

**Inheritance**

Inheritance is where one class becomes an extension of another class, therefore inheriting the members of that class.

There are two parties involved in an inheritance — a parent and a child. The parent is known as the superclass, sometimes also referred to as the parent class or base class. The child is known as the subclass or sometimes referred to as the child class or the derived class.

**In this, the child class inherits the members of the parent class** allowing classes to reuse data that already exists within other classes

For example, if we later decide to create an Employee class, then the Employee would also need a name, age, and gender.

**Instead of defining all of that all over again, we could have the Employee class inherit from the Person class that has defined the age, name, and gender of the person.**

When this happens, the Employee class now has the members of the Person class, and the programmer can focus on only defining things that are specific to the Employee class.

An inheritance relationship is known as a **“is a”** relationship.

Let’s try this in an example with a pseudo-code with Person as Superclass and Employee as Subclass:

**CLASS Person**

* PRIVATE STRING name
* PRIVATE INTEGER age
* PRIVATE STRING gender

FUNCTION getName()

* RETURNS STRING RETURN name

FUNCTION setName(STRING name)

* SET this.name = name

FUNCTION getAge() RETURNS INTEGER

* RETURN age

FUNCTION setAge(INTEGER age)

* SET this.age = age

FUNCTION getGender() RETURNS STRING

* RETURN gender

FUNCTION setGender(STRING gender)

* SET this.gender = gender

END CLASS

**Employee Class: Next, make a subclass of employees that inherits from the Parent class Person:**

CLASS Employee EXTENDS Person

* PRIVATE STRING employeeId
* PRIVATE STRING title

FUNCTION getEmployeeId() RETURNS STRING

* RETURN employeeId

FUNCTION setEmployeeId(STRING employeeId)

* SET this.employeeId = employeeId

FUNCTION getTitle() RETURNS STRING

* RETURN title

FUNCTION setTitle(STRING title)

* SET this.title = title

END CLASS

**Next, we test our work in the Inheritance tester class**

CLASS InheritanceTester

FUNCTION main(STRING[] args)

DECLARE Person person AS NEW Person

DECLARE Employee employee AS NEW Employee

END CLASS

**CHAPTER 9-B: Constructors in Inheritance**

Just to reiterate the rules about constructors in inheritance:

* A superclass's constructor is run before the subclass's constructor.
* Explicit calls can be made to a superclass's constructor from one of the subclass's constructors by using super ().
* Explicit calls to the superclass's constructor must be the first statement in the subclass's constructor.
* If the superclass does not have a default constructor, the subclass must explicitly call one of its other constructors.

**CHAPTER 9-C: Overriding and Overloading**

As we know a subclass inherits the members of its parent, however, a subclass may want to change the functionality of a method that it inherited.

This is allowed by ***overriding*** the inherited method.

**Overriding Inherited Methods**

For example, a square is a rectangle, meaning we could have a Square class that inherits from a Rectangle class. And let's say one of the methods that was inherited was to calculate the perimeter.

Well, the formula for calculating the perimeter of a square is different than the formula for calculating the perimeter of a rectangle.

So, while it's great that the Square class inherits this method, that class needs to change the behavior of that method. This is called overriding a method.

Let’s try this with a pseudo-code draft:

**Pseudo code for rectangle:**

CLASS Rectangle

PROTECTED double length

PROTECTED double width

PROTECTED double sides = 4

METHOD getLength RETURNS double

RETURN length

METHOD setLength(INPUT double length)

SET this.length = length

METHOD getWidth RETURNS double

RETURN width

METHOD setWidth(INPUT double width)

SET this.width = width

METHOD getSides RETURNS double

RETURN sides

METHOD setSides(INPUT double sides)

SET this.sides = sides

METHOD calculatePerimeter RETURNS double

RETURN (2 \* length) + (2 \* width)

METHOD public void print(){

System.out.println("I am a rectangle");

}

END CLASS

**Pseudo code for Square:**

CLASS Square EXTENDS Rectangle

METHOD calculatePerimeter RETURNS double

RETURN sides \* length

METHOD public void print(String what){

System.out.println("I am a " + what);

}

END CLASS

**Now make an inheritance tester:**

CLASS InheritanceTester

METHOD main(INPUT String[] args)

CREATE an instance of Rectangle called rectangle

CALL rectangle.print() // Assumes a print method exists in Rectangle

CREATE an instance of Square called square

CALL square.print("square") // Assumes a print method exists in Square

END CLASS

**CHAPTER 9-D: Limiting access to inheritance**

Constructors are not technically members of a class and therefore they are not inherited.

All the public and protected methods and fields in a superclass, those are indeed inherited, but the private methods and fields are not.

Also, if there are any final methods, meaning methods that have the word final in the header, these are inherited but cannot be overridden.

**EXERCISE:**

* The first one is going to be a superclass called Cake and it will have two fields, flavor and price. Go ahead and make getter and setter methods for both of these fields.
* Next create a BirthdayCake class which will inherit from the Cake class and it has a field of its own called candles. Go ahead and include the getter and setter method for candles.
* And then create a third class called WeddingCake which also inherits from Cake and has a field called tiers and you'll include a getter and setter method for this.
* And then finally, a class called TasteTester which will test out your inheritance.

**NOTE: Code shared on Git Hub.**

Let’s make a pseudo-code format for the following programs.

CLASS Cake

PRIVATE double cakeprice

PRIVATE String cakeflavor

CONSTRUCTOR Cake(INPUT String cakeflavor, INPUT double cakeprice)

CALL setCakeflavor(cakeflavor)

CALL setCakeprice(cakeprice)

METHOD getCakeflavor RETURNS String

RETURN cakeflavor

METHOD setCakeflavor(INPUT String cakeflavor)

SET this.cakeflavor = cakeflavor

METHOD getCakeprice RETURNS double

RETURN cakeprice

METHOD setCakeprice(INPUT double cakeprice)

SET this.cakeprice = cakeprice

END CLASS

**SUBCLASS Birthday Cake:**

CLASS BirthdayCake EXTENDS Cake

PRIVATE int candles

CONSTRUCTOR BirthdayCake

CALL super("Vanilla", 1200)

METHOD getCandles RETURNS int

RETURN candles

METHOD setCandles(INPUT int candles)

SET this.candles = candles

END CLASS

**SUBCLASS Wedding Cake:**

CLASS WeddingCake EXTENDS Cake

PRIVATE int tiers

CONSTRUCTOR WeddingCake

CALL super("Strawberry", 2000)

METHOD getTiers RETURNS int

RETURN tiers

METHOD setTiers(INPUT int tiers)

SET this.tiers = tiers

END CLASS

**Test Class:**

CLASS Test

METHOD main(INPUT String[] args)

* // Create a regular Cake object
* CREATE an instance of Cake called cake with flavor "chocolate" and price 3000
* PRINT "Cake flavor: " + CALL cake.getCakeflavor()
* PRINT "Cake price: " + CALL cake.getCakeprice()
* // Create a BirthdayCake object
* CREATE an instance of BirthdayCake called birthdayCake
* PRINT "Birthday cake flavor: " + CALL birthdayCake.getCakeflavor()
* PRINT "Birthday cake price: " + CALL birthdayCake.getCakeprice()
* // Create a WeddingCake object
* CREATE an instance of WeddingCake called weddingCake
* CALL weddingCake.setCakeflavor("pina colada")
* PRINT "Wedding cake flavor: " + CALL weddingCake.getCakeflavor()
* PRINT "Wedding cake price: " + CALL weddingCake.getCakeprice()

END CLASS