Method Override and Reuse

Lesson 3.2

Learning Outcomes

LO 3.2.1 **Reuse** parent class attributes/methods using the super keyword

LO 3.2.2 **Override** methods from parent classes

Inheritance

To apply inheritance from a superclass to a subclass in Java, you use the **extends** keyword.

The basic syntax of inheriting from a superclass to a subclass:

public class <subclass> extends <superclass> {…}

Example:

```
public class Customer extends Person {…}

public class Weapon extends Equipment {…}
```

Inheritance

When inheritance is applied to classes, all attributes and methods from the superclass *exists automatically* in the subclass even without physical implementation.

Even though it exists, it will still follow the scopes of *modifier access*.

If an attribute or method is declared as private in the superclass, since it can only be accessed in the same class, the class inheriting it cannot access it at any point of its own implementation.

Inheritance

```
public class Animal
{
    private String name;
    private int numFeet;

    public Animal(String name, int numFeet)
    {
        this.name = name;
        this.numFeet = numFeet;
    }
    public String getName()
    {
        return this.name;
    }
    public int getNumFeet()
    {
        return this.numFeet;
    }
}
```

```
public class Cat extends Animal
{
    private String breed;
    public Cat(String name, String breed)
    {
        this.name = name;
        this.numFeet = 4;
        this.breed = breed;
    }
    public String getBreed()
    {
        return this.breed;
    }
}
```

Inheritance

```
public class Cat extends Animal
{
   private String breed;
   public Cat(String name, String breed)
   {
       this.name = name;
       this.numFeet = 4;
       this.breed = breed;
   }
   public String getBreed()
   {
       return this.breed;
   }
}
```

Since name and numFeet are declared as *private*, only within the scope of class Animal can they be accessed therefore the current implementation will cause a compilation error.

Inheritance

```
public class Cat extends Animal
{
   private String breed;
   public Cat(String name, String breed)
   {
       this.name = name;
       this.numFeet = 4;
       this.breed = breed;
   }
   public String getBreed()
   {
       return this.breed;
   }
}
```

There are 2 ways this problem can be resolved:

1. By changing its modifier access
2. By assigning the values indirectly through the constructor of the superclass

protected Keyword

The scope of the keyword **protected** is shown in the table below.

In UML class diagram, it is denoted by the symbol, **#**.

	private	**public**	**protected**
Same class	Yes	Yes	Yes
Subclasses	No	Yes	Yes
Classes in the same package	No	Yes	Yes
Not subclass or same class/package	No	Yes	No

super Keyword

The **super** keyword can be used like the *this* keyword to access an *attribute* or *method* from a superclass.

Example:

```
super.name      //attribute

super.walk()    //method

super()         //constructor
```

Invoking constructor of superclass

When invoking the constructor of the superclass when implementing the constructor of the subclass, it should be placed on the **topmost** part of the implementation.

Example:

```
public class Cat extends Animal
{
    ...
    public Cat(String name, String breed)
    {
        super(name, 4);
        this.breed = breed;
    }
    ...
}
```

```
public class Cat extends Animal
{
    ...
    public Cat(String name, String breed)
    {
        this.breed = breed;
        super(name, 4);
    }
    ...
}
```

Inheritance Solution #1

```java
public class Animal
{
    protected String name;
    protected int numFeet;
    public Animal(String name, int numFeet)
    {
        this.name = name;
        this.numFeet = numFeet;
    }
    public String getName()
    {
        return this.name;
    }
    public int getNumFeet()
    {
        return this.numFeet;
    }
}
```

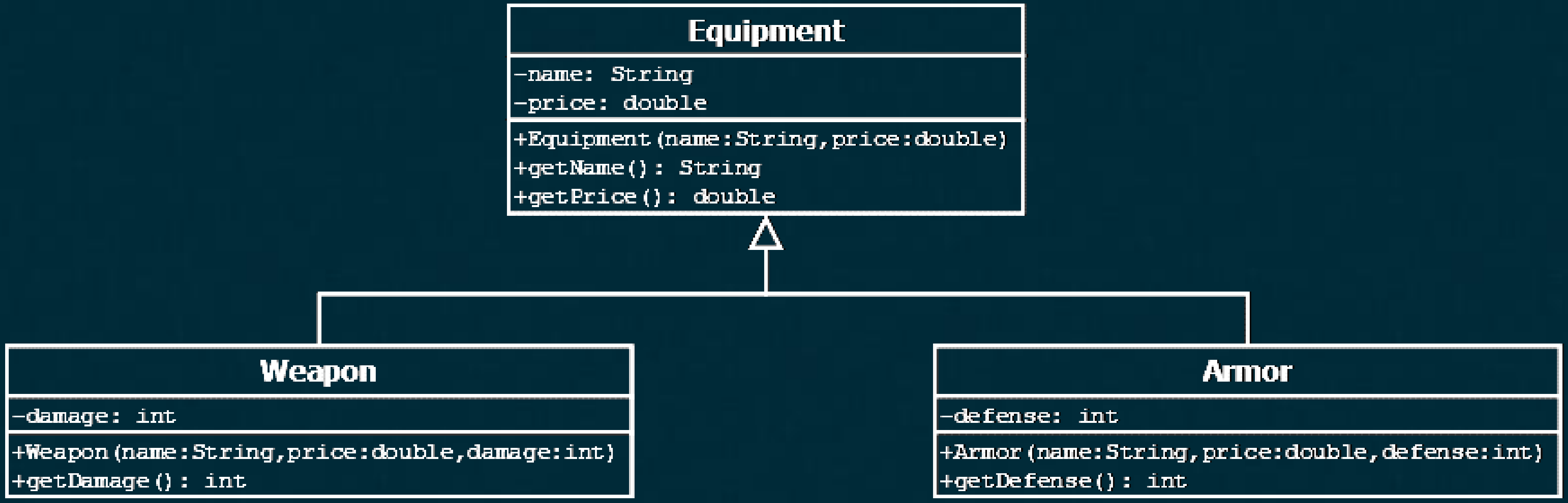
```java
public class Cat extends Animal
{
    private String breed;
    public Cat(String name, String breed)
    {
        super.name = name;
        super.numFeet = 4;
        this.breed = breed;
    }
    public String getBreed()
    {
        return this.breed;
    }
}
```

Inheritance Solution #2

```java
public class Animal
{
   private String name;
   private int numFeet;

   public Animal(String name, int numFeet)
   {
       this.name = name;
       this.numFeet = numFeet;
   }

   public String getName()
   {
       return this.name;
   }

   public int getNumFeet()
   {
       return this.numFeet;
   }
}
```

```java
public class Cat extends Animal
{
   private String breed;

   public Cat(String name, String breed)
   {
       super(name, 4);
       this.breed = breed;
   }

   public String getBreed()
   {
       return this.breed;
   }
}
```

Inheritance Implementation



Inheritance

```
public class Equipment
{
   private String name;
   private double price;
   public Equipment(String name, double price)
   {
       this.name = name; this.price = price;
   }
   public String getName()
   {
       return this.name;
   }
   public double getPrice()
   {
       return this.price;
   }
}
```

Inheritance

```
public class Weapon extends Equipment
{
   private int damage;
   public Weapon(String name, double price, int damage)
   {
       super(name, price); this.damage = damage;
   }
   public int getDamage()
   {
       return this.damage;
   }
}
```

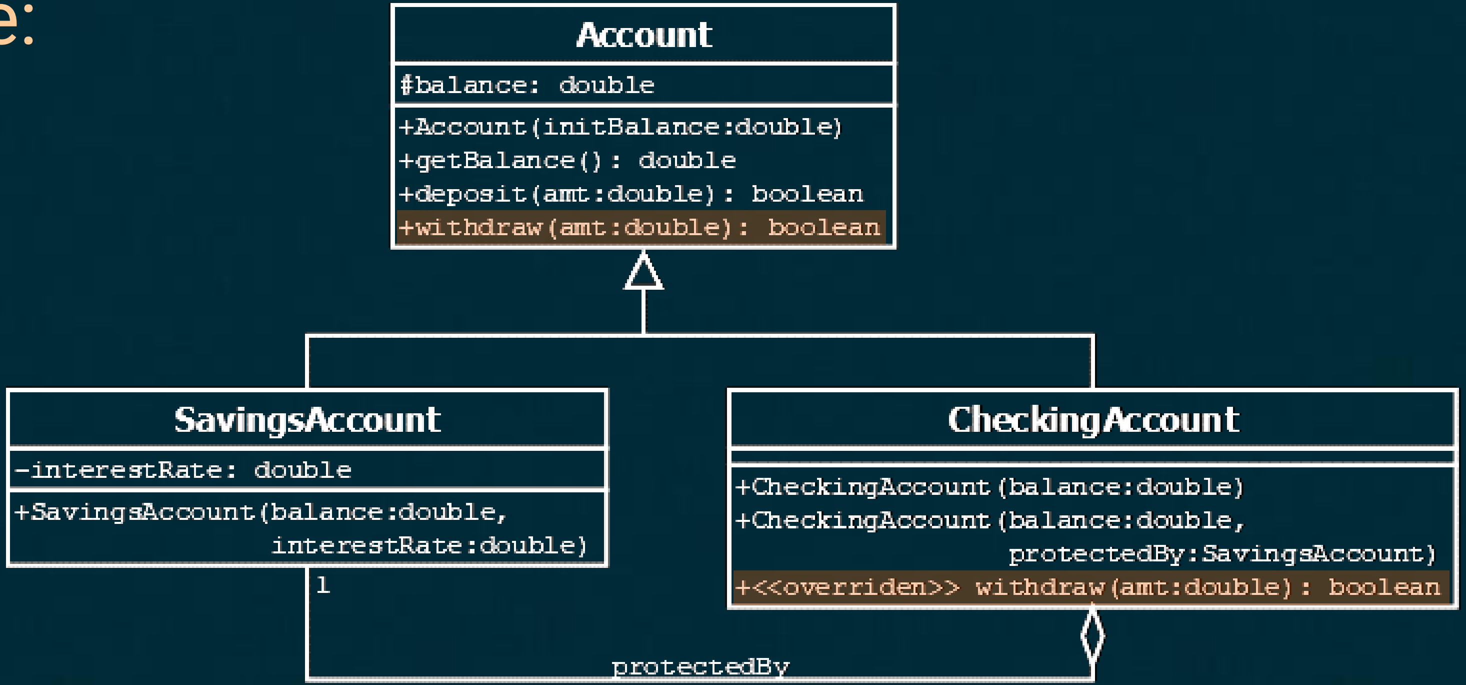
Inheritance

```
public class Armor extends Equipment
{
    private int defense;
    public Weapon(String name, double price, int defense)
    {
        super(name, price); this.defense = defense;
    }
    public int getDefense()
    {
        return this.defense;
    }
}
```

Overriding methods

If a method defined in a subclass such that the *name*, *return type*, and *argument list* exactly match those of a method in the superclass, then the new method is said to **override** the old one.

Example:


Account
#balance: double
+Account(initBalance:double)
+getBalance(): double
+deposit(amt:double): boolean
+withdraw(amt:double): boolean
SavingsAccount
-interestRate: double
+SavingsAccount(balance:double, interestRate:double)
CheckingAccount
+CheckingAccount(balance:double)
+CheckingAccount(balance:double, protectedBy:SavingsAccount)
+<<overriden>> withdraw(amt:double): boolean
1
protectedBy

Overriding methods

In design, overriding methods from a superclass is not mandatory. It is only executed when implementation of a method to a subclasses should be changed because either new attributes or procedure is integrated leading to a revised implementation of that superclass method.

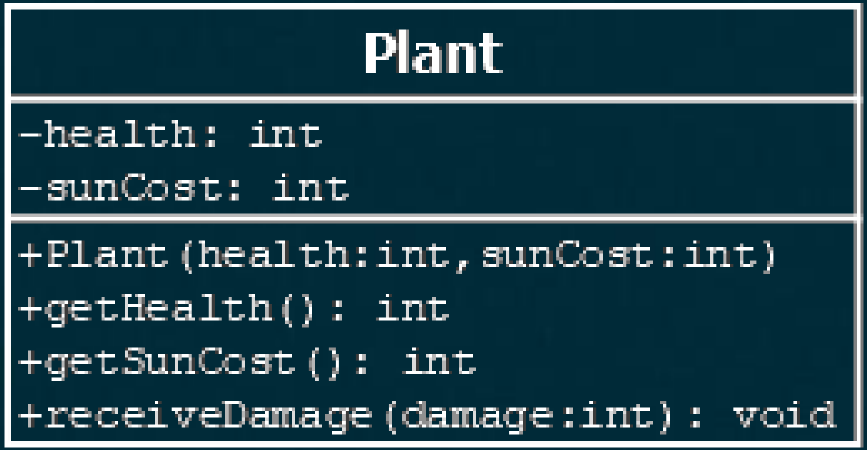
Overriding methods

A popular example would be the `toString` method of class Object. The superclass implementation of toString is in the form

```
<class name>@<hex representation of the object hash code>
```

But if you want the object to be represented in another form of a String value, you will override that method so that when the object is displayed, you can customize as to what values of the object should it be represented as.

LO 3.2.1 Reuse parent class attributes/ methods using the super keyword



```
Plant
-health: int
-sunCost: int
+Plant(health:int,sunCost:int)
+getHealth(): int
+getSunCost(): int
+receiveDamage(damage:int): void
```

Based on the game, ***Plants vs Zombies***, a class `Plant` is already implemented based on the UML class diagram on the left.

LO 3.2.1 Reuse parent class attributes/ methods using the super keyword

With Java code implementation below, create classes `SunProducer` and `Shooter`.

```java
public class Plant
{
    private int health;
    private int sunCost;

    public Plant(int health, int sunCost)
    {
        this.health = health;
        this.sunCost = sunCost;
    }

    public int getHealth()
    {
        return this.health;
    }
```

```java
    ...

    public int getSunCost()
    {
        return this.sunCost;
    }

    public void receiveDamage(int damage)
    {
        this.damage -= damage;
        if(this.damage < 0)
            this.damage = 0;
    }
}
```

LO 3.2.1 Reuse parent class attributes/ methods using the super keyword

A `SunProducer` is a Plant that varies in initial health, sun cost, and contains an attribute `sunPerProduction` that is assigned with a value only in object creation.

A `Shooter` is a Plant that initially has 300 health, varies in sun cost, and contains an attribute `shootingRate` that is also assigned with a value only in object creation.

All attributes must be assigned with a positive value, otherwise the value will be assigned to default value 1.

LO 3.2.2 Override methods from parent classes

With the current implementation of `Plant`, `SunProducer`, and `Shooter` classes, find a general method for these classes that can be overridden with any subclasses to-be-implemented by any of these 3 classes. Demonstrate method overriding by creating new subclasses and overriding that general method as necessary.