NC STATE UNIVERSITY

Inheritance and Polymorphism

Zybooks: 12.1-12.7

2

NC STATE UNIVERSITY

John Vlissides (GoF)

"I've said it before and I'll say it again: A hallmark – if not the hallmark – of good object-oriented design is that you can modify and extend a system by adding code rather than by hacking it. In short, ***change is additive, not invasive***."

3

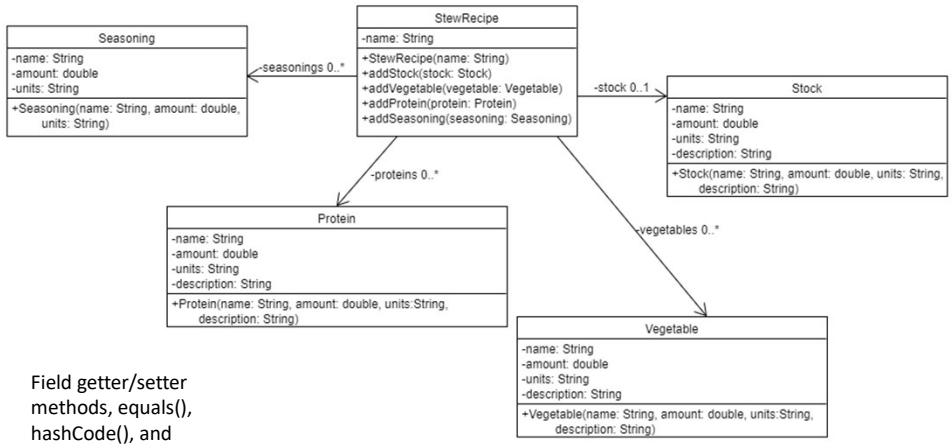
NC STATE UNIVERSITY

Code Reuse

- When writing programs, we want to write common code once and use it many times and in many contexts
  - Find commonalities
  - Customize with any additional information
  - Only one place to modify if there's a bug!

NC STATE UNIVERSITY

Stew Recipe Ingredient Commonalities



Field getter/setter methods, equals(), hashCode(), and toString() left out to simplify the diagram.

NC STATE UNIVERSITY

Inheritance

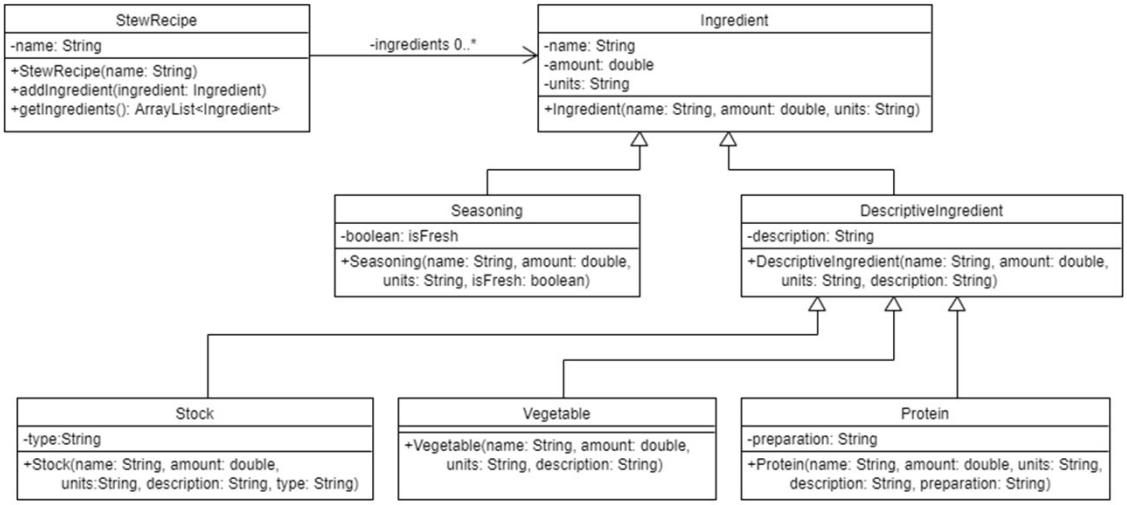
- Increases ability to reuse code
  - One class is an extension or specialization of another
  - Hierarchies of related object types
- Ingredients
  - All our ingredients have a common idea of *name, amount, and units*
  - Some ingredients have a *description* which may discuss how the ingredients are prepared (diced, chopped, cubed, etc.)

NC STATE UNIVERSITY

Is-a Relationship

- A hierarchical connection between two categories where one type can be treated as a ***specialized*** version of the other
- `Stock` is a specialized version of an `Ingredient`
- Inheritance Hierarchy: a set of hierarchical relationships between classes of objects.

Revised Stew Recipe Class Diagram



Definitions

- Inheritance (Inherit)
  - A programming technique in which a derived class extends the functionality of a base class, inheriting all of its state and behavior

- **Superclass**: the parent class in an inheritance relationship
- **Subclass**: the child class in an inheritance relationship

- Subclass *extends* the superclass

NC STATE UNIVERSITY

Single Inheritance

- In Java, a class can have ONLY ONE superclass

```
public class <Subclass> extends <Superclass> {

}
```

- A superclass may have many subclasses

10

NC STATE UNIVERSITY

Subclasses

- Are an instance of the super class
  - Superclass is constructed first
- Can have new state and add new behavior
  - Constructed/initialized after the superclass' state and behavior
- If we don't like the superclass' behavior, we can override it
  - To implement a new version of a method to replace code that would otherwise have been inherited from a superclass
  - Write the method you want to replace in the subclass
  - Method name and signature MUST match exactly

11

NC STATE UNIVERSITY

Construction with Inheritance (Incorrect)

```
public class Stock extends DescriptiveIngredient {
   private String type;
   public Stock(String name, double amount,
          String units, String description,
          String type) {
      this.name = name; //Set Ingredient field
      this.amount = amount; //Set Ingredient field
      this.units = units; //Set Ingredient field
      this.description = description; //Set DI field
      this.type = type; //Set Stock field
   }
}
```

This implementation requires that the instance fields from `DescriptiveIngredient` and `Ingredient` are public so that `Stock` can "see" the data.

Does this preserve the encapsulation of `DescriptiveIngredient` and `Ingredient`?

NC STATE UNIVERSITY

Redundancy (BAD!)

- Our implementation takes advantage of inheritance to describe all parent fields
  - By inheriting `DescriptiveIngredient` 
- To set superclass' fields, it requires parent classes' instance fields to be public/protected
  - Violates encapsulation 
- Each subclass constructor sets the parent classes' instance fields
  - Violates code reuse paradigm where a class should set its own fields 
- Each subclass could construct the parent type differently
  - Violates "one path of construction" paradigm 
- How could we fix?

Construction with Inheritance

- When constructing an ingredient, like `Stock`, you must
  - Construct the parent class (`DescriptiveIngredient`)
  - Which must in turn create its parent class (`Ingredient`)
  - And then construct the new information about `Stock`
- We want to ensure:
  - Each class controls its own construction
  - Protects its own encapsulation

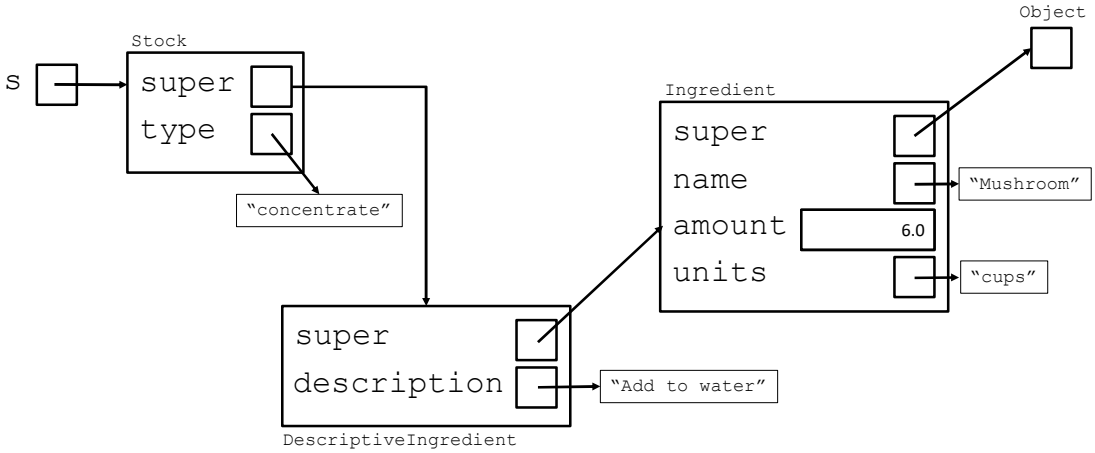
Interacting with the Superclass

- Subclass' constructors MUST call a superclass' constructor
  - Assumes that superclass has parameterless (default) constructor
  - Parameterless constructor is called automatically from subclass' constructor
  - If you don't have a parameterless constructor – compilation error!
- `super` keyword provides a reference to the behavior of the superclass
  - Use to call superclass constructor
  - Use to call superclass version of overridden methods

NC STATE UNIVERSITY

Constructing with Inheritance

```
Stock s = new Stock("Mushroom", 6, "cups", "Add to water", "concentrate");
```



16

NC STATE UNIVERSITY

Interacting with the Superclass

- Subclasses may not refer to private fields of the superclass
  - How do we access private fields/methods of a class? Through the public methods!
- Subclasses can call any of the public and protected methods and constructors of the superclass
  - Call a super constructor to create an instance of the super class.
- Constructors in the subclass need to accept the same parameters as the superclass' constructor in addition to parameters associated with specific data of the subclass
  - Parent parameters are passed to the super constructor.
- Constructors of superclass are NOT inherited
  - Want to continue principle of ONE path of construction & let each class control its construction.

17

Example Ingredient Implementation

```
public class Stock extends DescriptiveIngredient {
   private String type;
   public Stock(String name, double amount, String units,
          String description, String type) {
      super(name, amount, units, description);
      this.type = type;
   }
}
```

Call to super must be the first line in the subclass' constructor!

Always want to initialize the superclass first!

Overridden Methods

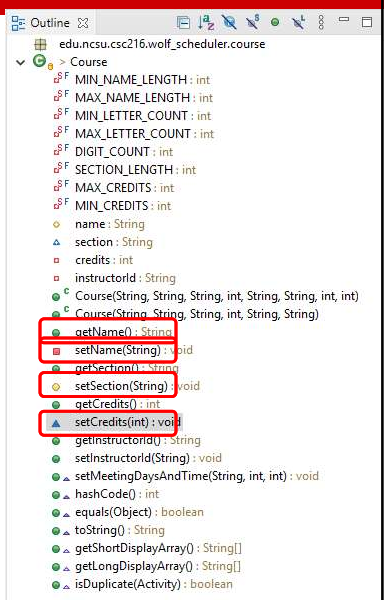
- If you override a method, and then want to call the superclass' version of the method, you must use **`super`** to call the method
  - Do this if you want to use a value from the super class in your overridden calculation (reduces redundancy)

```
//Override DescriptiveIngredient.getDescription() in
//Stock to include type
public String getDescription() {
   return super.getDescription() + " " + type;
}
```

NC STATE UNIVERSITY

Class Member Visibility

- There are 4 levels of visibility for members (fields and method) of a class
  - **Public** – everyone can see and use (green circle)
  - **Private** – only the class can see and use (red square)
  - **Protected** – the class, subclasses, and package members can see and use (yellow diamond)
  - **Default** (no modifier) – the class and package members can see and use (blue triangle)
- Outline = field/instance variable
- Filled = method



NC STATE UNIVERSITY

Class Member Visibility & Security

- There are 4 levels of visibility for members (fields and method) of a class
  - **Public** – everyone can see and use (green circle)
  - **Private** – only the class can see and use (red square)
  - **Protected** – the class, ***subclasses***, and package members can see and use (yellow diamond)
    - Visibility to subclasses is great! Visibility to package members is a ***security vulnerability!***
  - **Default** (no modifier) – the class and package members can see and use (blue triangle)

Vulnerability: anyone can create a package with the same name as yours! They can directly access the protected fields!

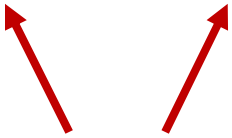
Workaround: make fields private and methods protected – that limits access and supports inheritance.

NC STATE UNIVERSITY

Reference Types

- When creating references and constructing objects, statements typically look like this:

```
Stock s = new Stock("Mushroom", 6, "cups", "Add to water", "concentrate");
```



The type on the left hand side of the operator should be the same as the type on the right hand side of the operator.

This is no longer always the case…

22

NC STATE UNIVERSITY

Polymorphism

- Legal for a variable of a superclass type to refer to an object of one of its subclasses

```
Ingredient i = new Stock();
```



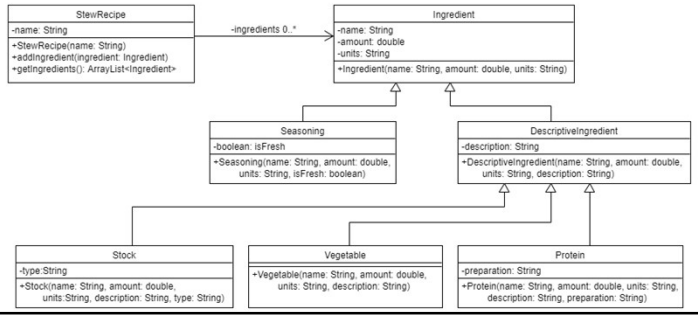
- Even though the variable is of type `Ingredient`, the object ***IS*** a `Stock` object, and it will behave like a `Stock` object. This means it uses `Stock`'s fields and methods!

23

NC STATE UNIVERSITY

Legal Statements?

- `Ingredient i1 = new Ingredient(…);`
- `Ingredient i2 = new DescriptiveIngredient(…);`
- `Ingredient i3 = new Stock(…);`
- `DescriptiveIngredient di1 = new Seasoning(…);`
- `DescriptiveIngredient di2 = new Stock(…);`
- `Stock s1 = new DescriptiveIngredient(…);`
- `Vegetable v = new Ingredient(…);`
- `Protein p = new Stock(…);`
- `Stock s2 = new Stock(…);`



NC STATE UNIVERSITY

Composition with Inheritance

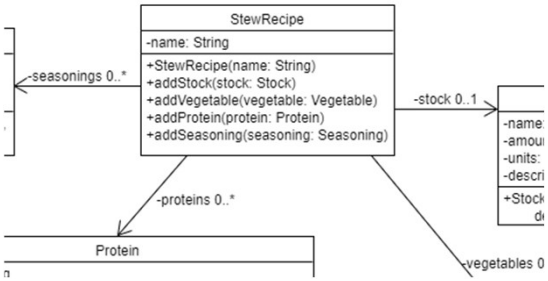
- The container class can use the parent type in the composition relationship
  - Takes advantage of polymorphism
- Therefore, our `Recipe` class can really contain an `ArrayList` of `Ingredients`
  - We can treat the ingredients polymorphically…
  - at least for the name, amount, and units.

NC STATE UNIVERSITY

StewRecipe Composition - Before

```
public class StewRecipe {
   private ArrayList<Seasoning> seasonings;
   private ArrayList<Protein> proteins;
   private ArrayList<Vegetable> vegetables;
   private Stock stock;
   private String name;

   public StewRecipe(…) {…}
   public void addStock(…) {…}
   public void addVegetable(…) {…}
   public void addProtein(…) {…}
   public void addSeasoning(…) {…}
}
```



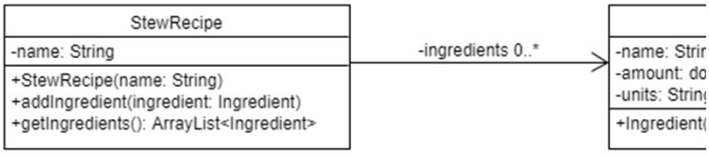
NC STATE UNIVERSITY

StewRecipe Composition - After

```
public class StewRecipe {
   private ArrayList<Ingredient> ingredients;
   private String name;

   public StewRecipe(…) {…}
   public void addIngredient(…) {…}
   public void getIngredients() {…}

}
```



NC STATE UNIVERSITY

Creating a StewRecipe

```
public class StewRecipeMain {
   public static void main(String [] args) {
      StewRecipe s = new StewRecipe("Chicken Chili");
      s.addIngredient(new Stock("Chicken", 3, "cups", "Add to water", "concentrate"));
      s.addIngredient(new Protein("Chicken", 1, "lb", "ground", "Brown in oil");
      s.addIngredient(new Vegetable("Onion", 1, "cup", "Diced");
      s.addIngredient(new Vegetable("Cannelloni beans", 2, "12 oz. can",
                                                    "Add after browning meat");
      s.addIngredient(new Vegetable("Tomatoes", 2, "12 oz. can",
                                                    "Add after browning meat");
      s.addIngredient(new Seasoning("Salt", 1, "Tbls.", false);
      s.addIngredient(new Seasoning("Cumin", 2, "Tbls.", false);
      s.addIngredient(new Seasoning("Garlic", 2, "Tbls.", true);

      System.out.println(s.printIngredients());
   }
}
```

What's printed?

NC STATE UNIVERSITY

Using Inheritance

- When should you use inheritance?
  - When you see similarities between classes that can be modeled in a hierarchy
- When should you NOT use inheritance?
  - When a subclass cannot be substituted everywhere a superclass could be used.

Has-a versus Is-a Relationships

- Has-a relationships are preferred over is-a relationships in cases where you class cannot or should not substitute for the other class
  - Most people choose to *have* and use software rather *be* a programmer

- **Unless an object MUST be an extension of another, use composition!**