NC STATE UNIVERSITY

Abstract Classes and Interfaces

Zybooks: 12.8-12.13

2

NC STATE UNIVERSITY

A Deeper Look at StewRecipe

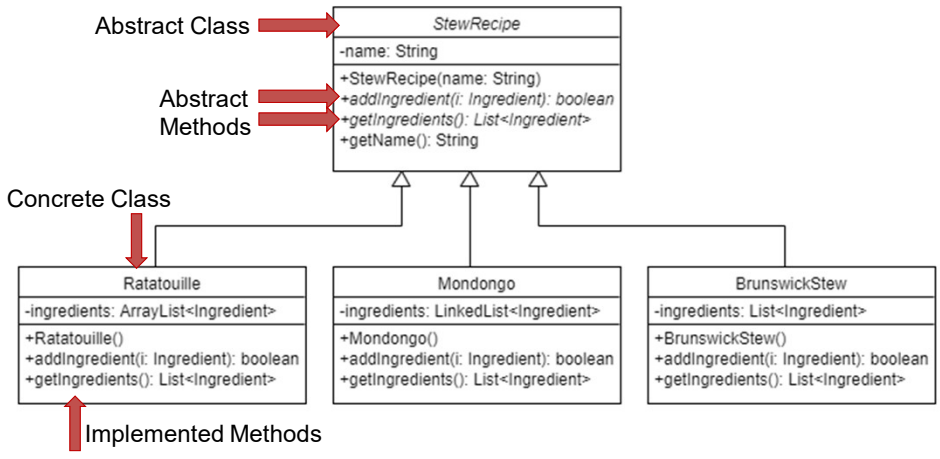
- Stew is a very general name for a "combination of solid food ingredients that have been cooked in liquid and served in the resultant gravy." [Wikipedia – Stew]
- There are stews from all around the world!
- Some stews might limit the types of ingredients allowed
  - Ratatouilles are a vegetable stew – no meat
  - Mondongos are made with beef tripe, pork's feet, and lot of vegetables
  - Brunswick Stews are tomato-based and has beans, vegetables, and chicken
- We can model the types of stews as an inheritance hierarchy and restrict the types of ingredients that can go in specific stews!

3

Abstract Classes

- A Java class that cannot be instantiated (directly constructed using the **`new`** keyword), but instead serves as a superclass to hold *common code* and declare *abstract behavior*
  - Want a class with common code (to reduce redundancy) but the class should never be instantiated (doesn't make sense in context of program)
  - **Abstract methods**: declared but not implemented – this is behavior that you promise to implement when you extend the abstract class with a concrete class
    - Concrete classes have implementation or code for all methods.
  - If every subclass of a superclass would override (provide custom behavior for) a particular method, consider making it abstract

StewRecipe Hierarchy



NC STATE UNIVERSITY

StewRecipe Class

- Make the **`addIngredient()`** method abstract
  - Different stews will each require different ingredients
  - Different stew recipes may also want to store their ingredients in different data structures
- Make the **`getIngredients()`** method abstract
  - If **`addIngredient()`** is abstract, then the data structure for storing ingredients has to be a field of the child class
  - **`getIngredients()`** can't be implemented in parent b/c the field is unknown.

NC STATE UNIVERSITY

StewRecipe (Abstract)

```
public abstract class StewRecipe {
   private String name;

   public StewRecipe(String name) {
      this.name = name;
   }

   public abstract boolean addIngredient(Ingredient i);
   public abstract List<Ingredient> getIngredients();

   public String getName() {
      return name;
   }
}
```

Abstract class keyword

Constructor that can only be called by child class' constructor(s).

Abstract method headers. End in ; to signify they have no implementation.

NC STATE UNIVERSITY

Abstract Classes

- Any class with abstract methods must be declared abstract in class header
- Abstract classes can have constructors to initialize its state
  - Concrete child classes should call those constructors to maintain paradigm of "one path of construction", but constructors cannot be called by client code.
- Cannot create instances of classes that are abstract

```
StewRecipe recipe = new StewRecipe("Ratatouille"); //legal?
```

- Classes without abstract methods can still be abstract
  - Make classes abstract when you don't want them instantiated by clients
- Concrete methods in abstract classes may call abstract methods, even if there is no implementation of the abstract method in the file.
  - Concrete subclass is required to implement the abstract methods
  - Can only call superclass method if there is an instantiated concrete object – with the overridden methods!
- Document like concrete classes – Javadoc!

NC STATE UNIVERSITY

Ratatouille Implementation (Concrete)

```
public class Ratatouille extends StewRecipe {
   private ArrayList<Ingredient> ingredients;
   public Ratatouille(String name) {
      super(name);
      ingredients = new ArrayList<Ingredient>();
   }
   public boolean addIngredient(Ingredient i) {
      if (i instanceof Protein) {
         return false;
      } else {
         ingredients.add(i);
         return true;
      }
   }
   public List<Ingredient> getIngredients() {
      return ingredients;
   }
}
```

Data structure for child.

Call to parent constructor as appropriate for inheritance.

Concrete implementation of abstract behavior.

NC STATE UNIVERSITY

Concrete Classes

- Concrete classes ***extend*** abstract classes and must implement abstract methods
  - Single inheritance still applies for the concrete classes
- Concrete class constructors still call one of the superclass' constructors
  - Pass appropriate parameters to superclass' constructor to initialize superclass' state
  - If no explicit call to a superclass constructor, try to go through the parameterless constructor

NC STATE UNIVERSITY

RecipeBook

```
public class RecipeBookMain {
   public static void main(String [] args) {
      ArrayList<StewRecipe> stewRecipes = new ArrayList<stewRecipe>();
      stewRecipes.add(new Ratatouille("Dr. Heckman's Ratatouille Recipe"));
      stewRecipes.add(new Mondongo("Dr. Domínguez's Mondongo Recipe"));
      stewRecipes.add(new BrunswickStew("Mr. Tetterton's Brunswick Stew"));
      Protein p = new Protein("Chicken", 1, "lb", "shredded", "grilled or roasted before shredding");
      for (int i = 0; i < stewRecipes.size(); i++) {
         //Which line is printed for each StewRecipe?
         if (stewRecipes.get(i).addIngredient(p)) {
            System.out.println("Chicken added to " + stewRecipes.get(i).getName());
         } else {
            System.out.println("Chicken NOT added to " + stewRecipes.get(i).getName());
         }
      }
   }
}
```

NC STATE UNIVERSITY

Pros and Cons of Abstract Classes

- Pros
  - Contains both concrete and abstract methods
    - Allows for common code
    - Provide common interface for functionality that all child classes should override
    - Guarantees functionality of abstract method(s) will exist when overridden
  - Allow for polymorphism
- Cons
  - Concrete classes can only extend one class (even if the extended class is abstract)

NC STATE UNIVERSITY

Drawbacks of Inheritance

- Java uses single inheritance
  - A class can only extend ONE class (only have ONE superclass)
  - Single inheritance prevents ambiguity if two parent classes share the same method signature(s)
  - Can't set up multiple is-a relationships
- What if we want an is-a relationship without sharing code?
  - We want to guarantee shared ***functionality*** of objects that are the same thing and the implementation is always dependent on the child
  - No common code, just common behaviors.

NC STATE UNIVERSITY

Interfaces

- A set of methods that classes promise to implement, allowing you to treat those classes as the interface type in your code
  - An interface is a contract that an implementing class MUST have the listed behaviors
- If an object implements an interface, the object has certain known behavior
  - Still can take advantage of polymorphism
- Classes can implement multiple interfaces

NC STATE UNIVERSITY

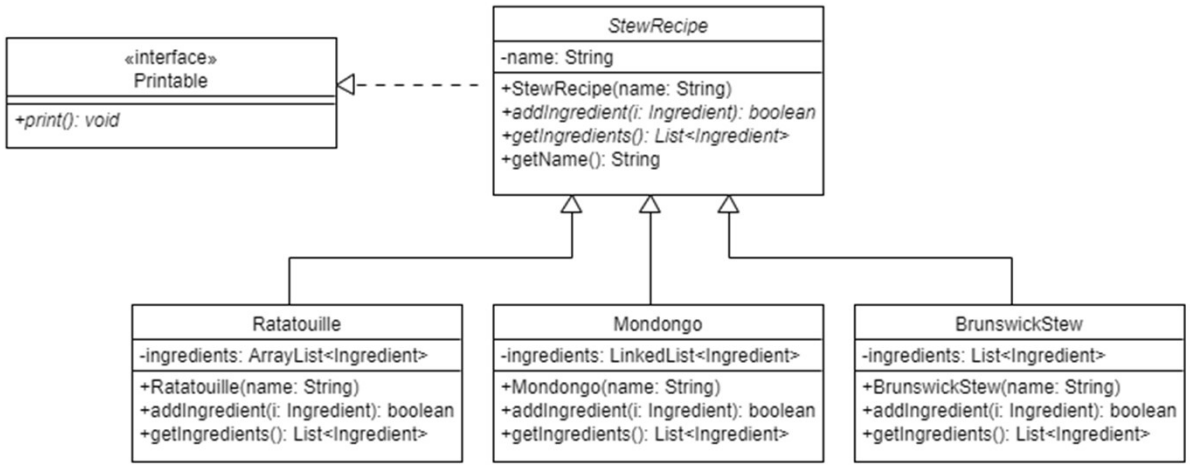
Interfaces (2)

- Interfaces define what something ***can do***
- A class defines what something ***is***

- **Example Interfaces**
  - `Runnable`: threads
  - `Comparable`: ordering and comparing objects
  - `Observable`: responding to UI events
  - `Serializable`: objects are saved to files
  - `List`, `Set`, `Map`, and `Iterator`: data structures

NC STATE UNIVERSITY

StewRecipe with Printable Interface



16

NC STATE UNIVERSITY

`Printable` Interface

Interfaces can also hold constants (public static final).

```
/**
 * Interface for things that can be printed
 * for publication
 */
public interface Printable {
    /**
     * Print the object for publication
     */
    void print();
}
```

Defined as interface; not class.

Interface methods are assumed public and abstract. Only need the return type, method name, and any parameters.

17

NC STATE UNIVERSITY

Interfaces

- Declared with keyword `interface`
- Only method signatures – no method code!
  - Methods are automatically `abstract`
- Don't need to use `public` on methods – all interface methods automatically `public`
- Interface methods cannot be `static`
- Data members are `public`, `static`, and `final` (constants)
- No constructors
- Document with Javadoc!

NC STATE UNIVERSITY

Revised StewRecipe Class

```
public abstract class StewRecipe implements Printable {
   private String name;
   public abstract boolean addIngredient(Ingredient i);
   public abstract List<Ingredient> getIngredients();
   public String getName() { return name; }
   public void print() {
      //print the recipe for publication
      System.out.println("I'm printing!!");
   }
}
```

List of interfaces implemented.

However, since StewRecipe is abstract, we could leave implementation of the interface's methods to the child classes of StewRecipe. Then each child can provide info on how to print the recipe in their context!

NC STATE UNIVERSITY

Interfaces and Abstract Classes

- Abstract classes (like `StewRecipe`) can implement interfaces which means all child classes automatically inherit the interface behaviors
  - Even better, you can choose NOT to implement the abstract method from the interface
  - Requires that the concrete child of the abstract class MUST implement the interface's behavior

NC STATE UNIVERSITY

Interfaces

- Interfaces can be used with other things not in the `StewRecipe` hierarchy
  - What else can you print?
- When thinking about designing interfaces in the projects, try to think about behaviors that could be used in other places beyond your project.
  - Another way of thinking about interfaces is as the definition of a set of behaviors that one part of your system is guaranteed to have.
  - Clients can then work with the interface type rather than the concrete type

NC STATE UNIVERSITY

RecipeBook Revisited

```
public class RecipeBookMain {
   public static void main(String [] args) {
      ArrayList<StewRecipe> stewRecipes = new ArrayList<stewRecipe>();
      stewRecipes.add(new Ratatouille("Dr. Heckman's Ratatouille Recipe"));
      stewRecipes.add(new Mondongo("Dr. Domínguez's Mondongo Recipe"));
      stewRecipes.add(new BrunswickStew("Mr. Tetterton's Brunswick Stew"));

      for (int i = 0; i < stewRecipes.size(); i++) {
          stewRecipes.print(); // print out the instructions in our recipe book
                               // for each recipe!
      }
   }
}
```


NC STATE UNIVERSITY

Benefits of Interfaces

- Implementing a common interface provides a type hierarchy similar to inheritance
  - Smart, uniform design
- Use to achieve polymorphism
  - All StewRecipes have a common set of functionality
    - Printable
- Example of additive not invasive
- Bi-pass limitations of single inheritance
- Specifies behaviors you want your class to have!
  - Think "something-able"

Legal Statements - Revisited

```
public interface I { void i(); }

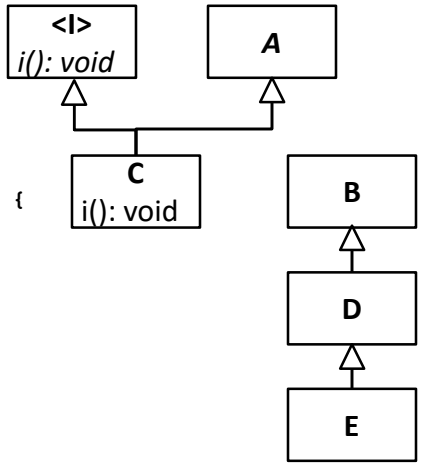
public abstract class A { ... }

public class B { ... }

public class C extends A implements I {
    public void i() { ; }
}

public class D extends B { ... }

public class E extends D { ... }
```




```
A a = new C();

I i = new C();

I id = new I();

A as = new A();

B b = new E();

D d = new D();

E e = new D();

E e = (E)new D();
```


Characterizing Inheritance

	Concrete Class	Abstract Class	Interface
Can be used as a reference variable type			
Can be instantiated (created directly with “new” keyword)			
Can have mutable fields (state)			
Can have method signatures with no implementation			
A class can extend/implement *one or many* of these?			

NC STATE UNIVERSITY

Testing with Inheritance

- Test concrete child classes like normal
  - If all classes in the hierarchy are concrete, test at all levels. Especially overridden behaviors.
- You cannot test abstract classes directly. You MUST test through a concrete child.
- Interfaces don't require testing – you test the actual implementations in a concrete class

NC STATE UNIVERSITY

The `Object` Methods and Inheritance

- `Object` is a (direct or indirect) superclass of all Java classes

Method	Purpose
`String toString()`	Returns a String representation of the object.
`boolean equals(Object obj)`	Indicates whether some other object is "equal to" this one.
`int hashCode()`	Returns a hash code value for the object.
`Class<?> getClass()`	Returns the runtime class of this `Object`

NC STATE UNIVERSITY

Generalizing `toString()` in Superclasses

- To make `toString()` useful by subclasses

```
@Override
public String toString() {
   return getClass().getName() + //info on fields;
}
```

- `getClass()` is a method of `Object` that returns a `Class` object. The `getName()` method returns the name of the class.
- The `toString()` on a child class should also be overridden to include info on the child's fields – use `super.toString()` to access the parent.

NC STATE UNIVERSITY

`equals()` and `hashCode()` with Inheritance

- Always handle the superclass first!

```
public class DescriptiveIngredient {
   public boolean equals(Object o) {
      //Test the superclass for equality first!
      if (!super.equals(o)) return false;
      if (getClass() != o.getClass()) return false;
      DescriptiveIngredient f = (DescriptiveIngredient)o;
      //test fields
   }
}
```

References

- Dr. Jo Perry's lecture notes