

CS456 Homework Assignment 7

Due date: _____

In the real world, your ideas and your work must be communicated effectively to your boss, to your client, and to potential investors.

Therefore, **please use clear and well-written English for your answers (as if I was your boss)**. **Incidentally, your boss expects typewritten reports, and so do I.** Drawings should be neatly done with an appropriate software program.

Problem #1: Consider the following code fragment.

```
public class EcoSimulator
{
    public void runSimulation(String ecoType)
    {
        Tree tree = null;
        Ground ground = null;

        if(ecotype.equals("coastal"))
        {
            tree = new PalmTree();
            ground = new Sand();
        }
        else if(ecotype.equals("forest"))
        {
            tree = new Pine();
            ground = new Dirt();
        }
        ...etc...
    }
}
```

Draw the UML for this code. Now refactor and draw improved UML. Now show me the relevant new code for your improved UML. (See class notes for an example of what I mean by “relevant” code.)

Problem #2: Refactor the following code for a sandwich-building robot. Include both the UML and the new code and justify your changes. As always, minimize changes to the existing code so that you minimize impacts on QA (but you will have to significantly change at least a few classes).

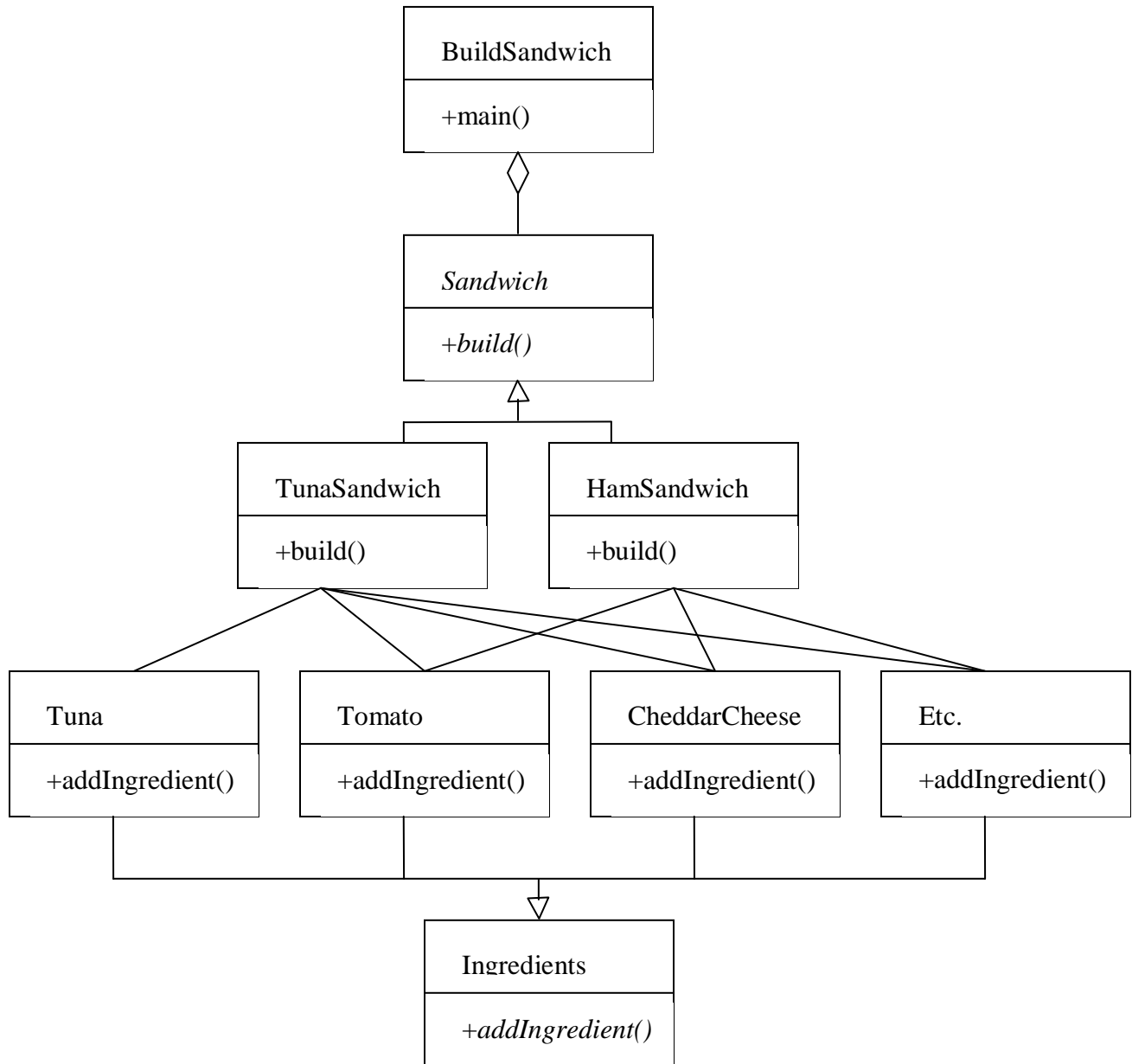
Hint: You may find that aggregation is a problem. Just remember that aggregation does not *always* mean that the object is passed into the class by a method or constructor. See my online notes for an example of how to aggregate without passing in the object

Note that the original UML does not contain RyeBread. To keep the design simple to understand, developers sometimes only show key classes. However, **all** classes that are affected by your refactoring **must** be included in the new UML. This applies to any job, anywhere.

You only need a code outline, similar to what is below. In other words, if I used a “//...” to indicate code details, I don’t expect you to fill in those details. The code below is online if you wish to copy it. You do not need to write a design document. Please see homework #2 and your class notes for additional info on the sandwich robot.

Does your code compile? It should! It may not do very much (because of the “//...”), but be sure to test it.

Sandwich-building robot:



The code:

```
/**
 * This builds the sandwich.
 */
public class BuildSandwich
{
    public static void main(String[] args)
    {
        //Note that I use inheritance so that I can refer to
        //the parent class. Only the instantiation has to change
        //if I want a new and different kind of sandwich.
        //The call to build() is unchanged because it is
        //common to all sandwiches.
        Sandwich mySandwich = new TunaSandwich();
        mySandwich.build();
    }
}

/**
 * Sandwich is abstract so that all types of sandwiches will
 * contain the build method. Very easy to add a new type of
 * sandwich by extending this class.
 */
public abstract class Sandwich
{
    /**
     * Builds the sandwich based on the sandwich ingredients.
     */
    public abstract void build();
}

/**
 * Here's my sandwich made of tuna on rye with
 * tomato, lettuce, and cheese.
 *
 * I extend the Sandwich class, so I have to implement
 * the abstract build method.
 */
public class TunaSandwich extends Sandwich
{
    /**
     * Note that I just create each ingredient and then
     * tell the robot to add the ingredient.
     *
     * How do I tell the robot to add the ingredient?
     * The method addIngredient is in *all* of the
     * ingredient classes because they all inherit
     * from the abstract Ingredient. So in each
     * ingredient class (like Tomato, Lettuce, or Tuna), I
     * give instructions on how to "spread", "lay down", or
     * otherwise handle that ingredient.
     *
     * Ah, that's the power of OO code!
     */
}
```

```

public void build()
{
    Ingredient ingredient = new RyeBread();
    ingredient.addIngredient();

    ingredient = new Tuna();
    ingredient.addIngredient();

    ingredient = new Tomato();
    ingredient.addIngredient();

    ingredient = new Lettuce();
    ingredient.addIngredient();

    ingredient = new CheddarCheese();
    ingredient.addIngredient();

    ingredient = new RyeBread();
    ingredient.addIngredient();
}

/**
 * Ingredient is abstract so that all sandwich ingredients will be
 * required to include the addIngredient method. i.e., all
 * ingredients will have this method in common.
 */
public abstract class Ingredient
{
    /**
     * Adds the ingredient using an appropriate technique.
     * e.g., spread with knife, lays down, or whatever.
     * Each ingredient will specify its own appropriate technique.
     */
    public abstract void addIngredient();
}

/**
 * All breads will have the same technique for adding this
 * ingredient ("lay down a slice). So I implement the addIngredient
 * method and inherit from this class to get specific types of bread.
 *
 * Note, I made Bread abstract so that it can't be instantiated
 * directly. Only the classes that inherit from it will be
 * instantiated (e.g., RyeBread, WheatBread, etc.).
 */
public abstract class Bread extends Ingredient
{
    /**
     * Lay down the slice of bread.
     */
    public void addIngredient()
    {
        //...
    }
}

```

```

/**
 * Contains any methods and variables specific to
 * rye bread. Don't need to implement addIngredient
 * because that was done in the bread class.
 */
public class RyeBread extends Bread
{
    //...
}

/**
 * Contains any methods and variables specific to
 * wheat bread. Don't need to implement addIngredient
 * because that was done in the bread class.
 */
public class WheatBread extends Bread
{
    //...
}

/**
 * This class extends Ingredient, so we have to implement the
 * method addIngredient. In this case we would include code
 * that directs the robot to spread the tuna.
 *
 * Note that I might have made all meats extend from a
 * common base class called Meat. I should do this if Tuna,
 * Ham, etc. have enough in common to warrant a common
 * base class that contains their common info. For simplicity,
 * I did not do this here.
 */
public Tuna extends Ingredient
{
    /**
     * Instructions to spread the tuna.
     */
    public void addIngredient()
    {
        //...
    }
}

/**
 * This class extends Ingredient, so we have to implement the
 * method addIngredient. In this case we would include code
 * that directs the robot to lay down the ham.
 */
public Ham extends Ingredient
{
    /**
     * Instructions to "lay down" the slice of ham.
     */
    public void addIngredient()
    {
        //...
    }
}

```

```

/**
 * This class extends Ingredient, so we have to implement the
 * method addIngredient.
 *
 * Might also have inherited from a common Vegi class, if
 * warranted.
 */
public Tomato extends Ingredient
{
    /**
     * Lay down tomato slice.
     */
    public void addIngredient()
    {
        //...
    }
}

/**
 * This class extends Ingredient, so we have to implement the
 * method addIngredient.
 *
 * Might also have inherited from a common Vegi class, if
 * warranted.
 */
public Lettuce extends Ingredient
{
    /**
     * Lay down lettuce leaf.
     */
    public void addIngredient()
    {
        //...
    }
}

/**
 * This class extends Ingredient, so we have to implement the
 * method addIngredient.
 *
 * Might also have inherited from a common Cheese class, if
 * warranted.
 */
public CheddarCheese extends Ingredient
{
    /**
     * Lay down cheese slice.
     */
    public void addIngredient()
    {
        //...
    }
}

/**

```

```

* A sandwich made of ham and cheese.
*
* I extend the Sandwich class, so I have to implement
* the abstract build method.
*/
public class HamAndCheeseSandwich extends Sandwich
{
    /**
     * Notice how easy it is to add whatever ingredients I want.
     * Simplicity enhances software engineering by making
     * testing and debugging easier. i.e., more reliability.
     */
    public void build()
    {
        Ingredient ingredient = new WheatBread();
        ingredient.addIngredient();

        ingredient = new Ham();
        ingredient.addIngredient();

        ingredient = new CheddarCheese();
        ingredient.addIngredient();

        ingredient = new WheatBread();
        ingredient.addIngredient();
    }
}

```

The code in main() would read:

```

Sandwich mySandwich = new HamAndCheeseSandwich();
mySandwich.build();

```