Tutorial 5 – Meta programming

Instructions

In this tutorial, you'll learn to apply Java's meta programming to the `PizzaDemo` application. In particular, you will use the reflection and annotation features to annotate and write input validation code for the `Pizza` classes of this application. You are given the `tutes.meta` package as starter code. You have to complete the following tasks:

1. Use the `DomainConstraint` annotation class to define some suitable domain constraints for the `Pizza` attributes. You also need to apply constraints to the `Topping` class attributes as well.

   For example, a pizza's size must be a valid string and the quantities and costs of these toppings must not be negative. Eg.

   ```
   @DomainConstraint(type = Type.String, optional = false, length = 15)
   String name;
   ```

2. The `DataManager` class in this Java code serves as a utility class which provides various static methods for handling data related to domain constraints and attributes of classes.

   The provided `DataManager` class comes with a method to create a new instance of a class from an array of arguments that are for a constructor method of that class. You do not need to implement this method, but you should take a look to see what it does.

   The method definition is:

   ```
   public static Object newInstance(Class c, Object[] attributeVals)
   throws NotPossibleException
   ```

   The parameter `attributeVals` is an array of attribute values whose types match the parameter types of a constructor of `c`. If a constructor cannot be found or cannot be invoked, this method will throw `NotPossibleException`. Otherwise, it returns a new object of `c` that is created by invoking the constructor method which takes the arguments specified in the `attributeVals` array.

   You should validate the input value of each attribute by invoking the `validateDomainAtttribute` method before using this method to create a new instance.

3. Implement the `validateDomainAttribute` method from the lecture in the `DataManager` class (see `app.MetaApp` in Lecture 5 source code) .

   The header of the method must be the following:

   ```
   public static validateDomainAttribute(Field f, Object value) throws
   NotPossibleException
   ```

   Note that this method throws a `NotPossibleException` immediately after a failed validation.

   In class `DataManager`, you should implement these methods:

```
// return an attribute of c
private static Field getAttribute(Class<?> c, String name)
    throws NotPossibleException

// return the annotated attributes of a class
public static List<Field> getAttributes(final Class<?> c,
    final Class<DomainConstraint> annotatedClass)
    throws NotPossibleException

// validate a value against the domain constraint of an attribute
public static Object validateDomainAttribute(Field f, Object value)
    throws NotPossibleException

// validate multiple values against the domain constraint of
// multiple attributes of a class
public static void validateDomainAttribute(Class<?> c,
    Object[][] attributeValPairs)
    throws NotPossibleException
```

4. Implement another input validation method which you will use later in the `PizzaDemo` application. It is used as a shortcut to the method in the previous step. The method header is as follows:

   ```
   public static void validateDomainAttribute(Class c, String name,
       Object value)
       throws NotPossibleException
   ```

   This method should find the attribute of the class `c` that has the given name and invoke the method in the previous step to validate its value.

   ```
   // validate an input value of an attribute of a class against
   // the attribute's domain constraint
   public static Object validateDomainAttribute(Class<?> c,
       String attribute, Object value)
       throws NotPossibleException {
         Field f = getAttribute(c, attribute);
         return validateDomainAttribute(f, value);
   }
   ```

5. Update the `PizzaDemo` class so that the code that generate new toppings and pizzas must validate the data values before creating a new object. Further, object creation must be performed using the `DataManager.newInstance` method.

   Note: it is suggested that you create several static helper methods to create a new topping and a new instance of each type of pizza.

   ***** You need implemented Toping constructor to allows a user to pass in Double and Integer objects for cost and quantity.

   ```
   /**
    *  It allows a user to pass in Double and Integer objects for cost
   and quantity,
   ```

```
 *  respectively, instead of explicitly converting them to primitive
types.
 */
public Topping(String name, Double cost, Integer qty) {
    this(name, cost.doubleValue(), qty.intValue());
}
```