JP2 2020: Lab exam

Overview

Your task is to develop classes to model a world involving **Citizens** and **Traders**, where **Citizens** are able to exchange gems for **Goods** including bread, wool, armour, weapons, and building materials. All of those goods are provided by **Traders** in exchange for gems. Every **Trader** supports one or more **Trades**, where a **Trade** includes the following information:

- The price in gems
- The specific goods that are for sale
- How much of that goods is available for the price

The following is an example of a **Trade**:

- Trade **1 gem** for **3 Bread**

Every **Citizen** has an amount of gems along with an amount of Goods of each type (which may be zero). When a **Citizen** successfully executes a **Trade**, the following happens:

- The relevant amount of gems is removed
- The relevant amount of Goods is added

For example, if a **Citizen** initially has 5 gems and an empty inventory, after executing the above trade, they would have 4 gems and 3 Bread in their inventory. If a **Citizen** does not have enough gems to complete a trade, they cannot execute it.

Trades are provided by **Traders**. When a new **Trader** is created, they only have one Trade available. Each time a **Trader** makes a trade with a **Citizen**, a new (randomly-chosen) **Trade** is added to their set. So if a **Trader** starts with the sample **Trade** above, after three successful trades with **Citizens**, the **Trader** might have the following **Trades** in their list:

- Trade **1 gem** for **3 Bread**
- Trade **2 gems** for **1 Helmet**
- Trade **1 gem** for **1 Bread**
- Trade **2 gems** for **4 Wool**

(Note that this trader has two **Trades** for the same item of **Goods** – Bread. This is not a problem in the system – you do not need to check for duplicates.)

Task 1: Goods (2 marks)

*Note about implementation: all classes created in this exam should be put in the **trading** package.*

You must create an enumerated type **Goods** with the following values:

BREAD, COAL, FISH, HELMET, IRON, PAPER, SHIELD, SWORD, WOOD, WOOL

Task 2: Trade (6 marks)

You must create a class **Trade** representing a single trade, including the following properties:

- gems: the number of gems involved in the trade (an integer)
- amount: the amount of goods involved in the trade (an integer)
- goods: the type of goods involved in the trade (an object of type Goods)

For example, the trade "1 gem for 3 BREAD" would be represented as:

- gems: 1
- amount: 3
- goods: Goods.BREAD

The **Trade** class should have a public constructor that initialises all of the fields, and should also include the following methods:

- A complete set of **get** methods, but no **set** methods
- Appropriate implementations of **equals()** and **hashCode()** – note that equality should be based on the values of all three fields.
- An implementation of **toString()** that produces a string representation of the **Trade** similar to the example above ("1 gem for 3 BREAD").

Not that you can use automatically-generated code for the **get** methods and **equals()**/**hashCode()**, but you will need to write the **toString()** method by hand to meet the specification.

Task 3: Citizen (5 marks)

Next, create a class **Citizen** representing a citizen in the game. The internal details of this class are up to you; here is the required behaviour.

The constructor for **Citizen** should take a single parameter, an integer representing the number of gems, and should create a new **Citizen** with that many gems and an empty inventory.

Citizen should have the following **public** methods:

- **public int getGems()** – returns the current amount of gems
- **public int getAmount (Goods goods)** – returns the current amount of the indicated **Goods** type in the inventory. Should return 0 if the **Citizen** does not have any of the indicate **Goods.**
- **public boolean executeTrade (Trade trade)** – should check whether the given trade is possible (i.e., whether the **Citizen** has enough gems)
    - If the amount of gems is not enough, **return false** and do not change anything
    - Otherwise, update the amount of gems and the inventory based on the details of the Trade and **return true**

Task 4: Trader (3 marks)

Create a class **Trader** representing a trader in the world. As with **Citizen**, the internal details of this class are up to you; here is the required behaviour.

The constructor for **Trader** should take no parameters, and should create a **Trader** with a single, randomly-chosen **Trade**.

Trader should have the following **public** methods:

- **public List<Trade> getTrades()** – returns the current list of **Trades** supported by this **Trader**
- **public addRandomTrade()** – adds a new, randomly-chosen **Trade** to the list. The **Trade** should be generated with the following constraints:
    - The value for **gems** should be between 1 and 5 (inclusive)
    - The value for **amount** should be between 1 and 5 (inclusive)
    - The value for **Goods** should be randomly chosen from the values of the **Goods** enum

To generate a random number, you can use the **java.util.Random** class, as follows:

- Creating an object: **Random rand = new Random();**
- Later on in the code, when you need a number: **int value = rand.nextInt(n);** will return a number between 0 and (n-1), inclusive

Task 5: Trade.execute() (3 marks)

In your **Trade** class, implement one additional public method, as follows:

- **public void execute(Trader trader, Citizen citizen)**

This method should behave as follows:

- If the current **Trade** is not included in the list of trades supported by **trader**, this method should throw an **IllegalArgmentException**
- Otherwise, it should call **citizen.executeTrade()** with the current trade
    - If **executeTrade()** returns true, the method should also call **trader.addRandomTrade()**

Style (1 mark)

To get full marks for style, you should ensure the following:

- Every class file should have a Javadoc comment at the top describing the class
- Every non-trivial method should have an implementation comment
- Indentation and variable names should follow normal Java conventions

What to submit

You should upload the following four files:

- Goods.java
- Trade.java
- Citizen.java
- Trader.java

Be sure to submit to the correct submission link, and be sure to submit before the deadline.