Design Patterns

Name: _____ ID: _____

These questions do not have a formal, definitive answer. They are meant to be food for thoughts. Feel free to seek answers on browsing the Internet, talking to other software developers or reading books.

1. What is a design pattern? Can you think about any analogies to define them?

2. Ok, imagine that a beautiful day you wake up feeling like creating a design pattern. You call your best friend publisher, and writes a paper about your design pattern. Is this the way design patterns are created?

3. The *GoF* book classifies design patterns into three categories: *creational*, *structural* and *behavioral*. Define each of these categories.

4. The board of directors of *Toy Inc.* has shifted gears, and now is going towards the video-game market. They have hired you to develop a game: *Raiders of Alucubaca*. This is a virtual reality game, where each player controls an avatar, that will interact with other players, and with items in a fantasy world. Of course, we are not going to implement this game from beginning to the end. We will adopt a *bottom-up* approach. So, we will start with a small part of the whole application: the items that make up the virtual world. Items can be simple or composite. A simple item is something indivisible, like an arrow bolt, a precious gem or a rope. A composite item is made up of a number of other items, which can be either simple or composite. For instance, a bow is made of a little wood stick plus a string. A necklace is made by a chain plus some precious stones and adornments. An armor is made of a breast plate, straps and shoulder plates. Every item should implement the following interface:

```
public interface Item {
  double getValue();
  int getAge(); // in days
  double getWeight(); // in pounds
  int getHitPoints();
  void setDamage(int damage);
  String getName();
  String getDescription();
}
```

How would you go about handling this item affair? There is a structural design pattern called *Composite* that fits this problem perfectly. Use this pattern to implement items.

5. Constructing items can be a real pain. Imagine if, every time we need an instance of the priestess's necklace, we had to do:

```
CompositeItem c1 = new CompositeItem("Necklace", "The priestess's necklace");
c1.addItem(new Chain("Chain of gold"));
c1.addItem(new Ruby("The Tear of Blood"));
c1.addItem(new Diamond("The glittering raindrop"));
c1.addItem(new Diamond("The sparkling sun"));
return c1;
```

   There is a creational design pattern that is goes hand-in-hands with the composite pattern: this is the *Builder*. How could we use the builder pattern to factor the construction of complex items into closed pieces of code?

6. Tell me what will be printed by the `main` method in the program below:

```
public class Avatar {
  public void buy(Knife k) {
    System.out.println("Avatar bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Avatar bought a sword");
  }
}
```

```
public class Knife {}
```

```
public class Sword extends Knife {}
```

```
public class SpiderAv extends Avatar {
  public void buy(Knife k) {
    System.out.println("Spider bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Spider bought a sword");
  }
}
```

```
public static void main(String args[]) {
  Avatar a1 = new Avatar();
  Avatar a2 = new SpiderAv();
  SpiderAv sa = new SpiderAv();
  Knife k = new Knife();
  Sword s = new Sword();
  Knife ks = new Sword();
  System.out.println("a1");
  a1.buy(k);
  a1.buy(s);
  System.out.println("a2");
  a2.buy(k);
  a2.buy(s);
  System.out.println("sa");
  sa.buy(k);
  sa.buy(s);
  System.out.println("???");
  a1.buy(ks);
  a2.buy(ks);
  sa.buy(ks);
}
```

7. Java is a programming language that does not provide something called *double dispatch*. If it did, the output of the program above would certainly be different. What is double dispatch?

8. It is possible to simulate double dispatch in Java, and we will do this now. Add a method `void isBoughtBy(Avatar a)` to the classes `Knife` and `Sword`, so that a command like `item.isBoughtBy(avatar)` prints a result according to the dynamic type of `item` and `avatar`. What would be printed by the program below?

```
public static void main(String args[]) {
  Avatar a1 = new Avatar();
  Avatar a2 = new SpiderAv();
  SpiderAv sa = new SpiderAv();
  Knife ks = new Sword();
  ks.isBoughtBy(a1);
  ks.isBoughtBy(a2);
  ks.isBoughtBy(sa);
}
```

9. Coming back to *Raiders of Alucubaca*, if I run code like this:

```
Item treasure = (new TreasureBuilder()).getItem();
System.out.println(treasure);
```

My virtual machine will print something like:

```
CompositeItem@cafa9e
```

What is obviously not what I want. Code a way to print a nice description of the item, say:

```
Diamond: A glittering diamond
Chain: Chain of gold
Ruby: The Tear of Blood
Diamond: The Sparkling Star
```

10. Very good! Now we need to implement a program that simulates the *dealer*, which is the person who buys and sells items in *Raiders*. So, I would like to have a program to compute the value of an item. But you can't call the method `getValue` recursively, because we will be using the dealer's table, which pays much less, of course! How can we compute such a program?

11. We are doing good progress! Now, we need a program to discover when an item is damaged so badly that it cannot be used. This program must find the value of the item with the lowest amount of *hit points*. Remember, items have a method `int getHitPoints()`, which may be of some help. Are the similarities between this problem and the other two problems in this page? Can we do some code reuse here? What the *visitor* design pattern can do for us?

12. Now that *Raiders of Alucubaca* has become a huge (HUGE!!!) success, our servers are starting to have problems with the volume of data that must be stored. The main problem is exactly because of the items. Most of the items, like knifes, coins, pans and so forth are too similar, but still we keep different copies of all of them. How can we solve this? Can we use the *flyweight* design pattern in this case? Code a way to avoid redundancies in the creation of items.