Java Foundations Practices - Section 2

Problem 1: Simple ASCII Art

Overview: Using text to create a picture is known as ASCII art. In section 2, we made an ASCII art cat. In this practice, you'll use print statements to recreate the image above. Task Use 8 print statements to recreate the smiley face above. Your art will rely on only a single character, besides space, such as X or #. The ProblemSet2_1 project is available to help you get started

To create a smiley face using ASCII art with only a single character (like `#` or `X`), you can use `print` statements to build the image line by line. Here's a simple example using `#` for a smiley face:

```java
public class ASCIIArt {
    public static void main(String[] args) {
        // Print the smiley face using # character
        System.out.println("  #####  ");
        System.out.println(" #     # ");
        System.out.println("#  # #  #");
        System.out.println("#       #");
        System.out.println("#  # #  #");
        System.out.println(" #     # ");
        System.out.println("  #####  ");
    }
}
```

Explanation

1. **`System.out.println("  #####  ");`**: Prints the top of the smiley face.
2. **`System.out.println(" #     # ");`**: Prints the top row of eyes.
3. **`System.out.println("#  # #  #");`**: Prints the eyes and the space in between.
4. **`System.out.println("#       #");`**: Prints the face's middle section.

Java Foundations Practices - Section 2

5. **`System.out.println("#  # #  #");`**: Prints the mouth's upper part.

6. **`System.out.println(" #     # ");`**: Prints the bottom row of the face.

7. **`System.out.println("  #####  ");`**: Prints the bottom of the smiley face.

You can adjust the characters and spacing to fine-tune the look of your smiley face. Just make sure to use exactly 8 `print` statements to meet the requirement.

Problem 2: Original ASCII Art Overview Using text to create a picture is known as ASCII art. In section 2, we made an ASCII art cat. In this practice, you'll create your own beautiful work of art.

Task Use print statements to create your own beautiful original ASCII art. Use comments to describe what your image is depicting. It's ok for your art to rely on only a single character, besides space, such as X or #. But you're encouraged to use a few different characters in your design, like in the cat example from class. Your art must also:

- **Use at least 8 print statements**
- **Be at least 8 characters wide**
- **Use at least 20 characters that aren't space You're welcome to create another cat.**

However, this image must look significantly different from the example used in class. Similarly, you're welcome to create another face, but it must look significantly different from the face in the previous practice (it's way too easy to turn the smiley face into a frowny face). Note: The backslash (\) character has special meaning in Java print statements. If you choose to use a backslash in your image, you'll actually need to write two backslashes (\\) in your print statement. The ProblemSet2_2 project is available to help you get started

Here's an example of a different piece of ASCII art: a small dragon. This piece uses multiple characters and satisfies the requirements.

```java
public class ASCIIArt {
    public static void main(String[] args) {
        // This ASCII art depicts a small dragon
        System.out.println("        _______________ ");
        System.out.println("       /            \\ ");
        System.out.println("      /  /\\     /\\    \\ ");
```

Java Foundations Practices - Section 2

```
        System.out.println("     / / \\  / \\   \\ ");
        System.out.println("    / /   \\ /   \\   \\ ");
        System.out.println("   / /     \\     \\   \\ ");
        System.out.println("  / /       \\_____\\   \\ ");
        System.out.println(" / /         \\_____/    \\ ");
        System.out.println("/ /                 /   / ");
        System.out.println("/__/_____________________/____/");
    }
}
```

Explanation

1. **`System.out.println("        _____________ ");`**: Prints the top of the dragon's body.
2. **`System.out.println("       /           \\ ");`**: Prints the top part of the wings.
3. **`System.out.println("      / /\\    /\\   \\ ");`**: Prints the first part of the wings and the body.
4. **`System.out.println("     / / \\  / \\   \\ ");`**: Continues the wings and body.
5. **`System.out.println("    / /   \\ /   \\   \\ ");`**: Expands the body.
6. **`System.out.println("   / /     \\     \\   \\ ");`**: Further expands the body.
7. **`System.out.println("  / /       \\_____\\   \\ ");`**: Completes the body and starts the lower part of the wings.
8. **`System.out.println(" / /         \\_____/    \\ ");`**: Finishes the wings.
9. **`System.out.println("/ /                 /   / ");`**: Prints the lower body part.
10. **`System.out.println("/__/_____________________/____/");`**: Prints the base of the dragon.

This dragon is made of multiple characters (`/`, `\\`, `_`, and space) and meets all the requirements. Feel free to adjust the design or characters to suit your preferences!

Problem 3: The Snake Box Factory

Overview :Dear Respectable Software Engineer, Here at the world renowned Snake Box Factory, we pride ourselves on our ability to deliver the highest quality, custom sized, cardboard boxes to our customers. Our boxes are filled with the highest quality, custom-ordered snakes. We service thousands of accounts worldwide and have a solid 98% satisfaction rating with customers.

Java Foundations Practices - Section 2

However, the entire ordering process is currently written on cardboard, which is transported between departments via carrier snake. We thought this would be a good way to show confidence in the quality and usefulness of our product. But as our business continues to grow, we're realizing this was a bad idea. We believe it's time for a more conventional and digitized approach to our operations. Would you be able to help us develop the software we need to make this happen? Sincerely, President George Johnson, The Snake Box Factory Tasks Read the scenario found in the overview and consider what objects could be modeled as part of creating a software solution. Identify 3 objects from this scenario (remember, objects can be either tangible or abstract. List 3 properties and 3 behaviors belonging to each object. Write your solution as a document rather than a .java fil

Solution Document for Snake Box Factory Software

Overview:

The Snake Box Factory needs a software solution to manage the custom-sized cardboard boxes and the snakes inside them. To design this software, we'll model three main objects: `Box`, `Snake`, and `Order`. Each object will have properties and behaviors that reflect the requirements of the factory.

---

1. Box

Properties:

1. **Size**: Dimensions of the box (length, width, height).

2. **Material**: Type of cardboard or material used.

3. **Quantity**: Number of snakes that the box can contain.

Behaviors:

1. **CalculateVolume()**: Computes the volume of the box based on its dimensions.

2. **AddSnake(Snake snake)**: Adds a snake to the box, if space permits.

3. **CheckCapacity()**: Checks if the box has space available for more snakes.

Java Foundations Practices - Section 2

2. Snake

Properties:

1. **Species**: The type or species of the snake.
2. **Length**: The length of the snake.
3. **Color**: The color pattern or coloration of the snake.

Behaviors:

1. **Slither()**: Simulates the movement of the snake.
2. **Eat()**: Simulates the snake eating (if applicable).
3. **ChangeColor(String newColor)**: Changes the color of the snake to a new specified color.

---

3. Order

Properties:

1. **OrderID**: A unique identifier for each order.
2. **Box**: The box associated with the order.
3. **Snakes**: A list of snakes to be placed inside the box.

Behaviors:

1. **ProcessOrder()**: Handles the processing of the order, including preparing the box and adding snakes.
2. **VerifyOrder()**: Checks if the order details (such as box size and number of snakes) are valid.
3. **ShipOrder()**: Prepares the order for shipment, including packaging and labeling.

Conclusion:

Java Foundations Practices - Section 2

The proposed objects `Box`, `Snake`, and `Order` encapsulate the key aspects of the Snake Box Factory's operations. Each object has relevant properties and behaviors that will help in managing the factory's inventory and orders efficiently. This model provides a foundation for developing the necessary software to digitize and streamline the factory's processes.

Feel free to modify or expand upon these objects based on further requirements or specific needs of the Snake Box Factory.