# Lab Assignment

Name: Dudhatra Devanshi, Heet Dobariya
Roll no: 22BCP171, 22BCP177
Division: 3(G5)

## 1) Differences between Factory Design , Builder Design and Prototype Design are :

The Factory, Prototype, and Builder design patterns are all creational design patterns, meaning they deal with the process of object creation. However, they serve different purposes and are applied in different scenarios.

1. Factory Design Pattern:

- Intent: The Factory pattern is used to create objects without specifying the exact class of the object that will be created. It provides an interface for creating instances of a class, but the exact class of the object isn't determined until runtime.
- Usage: When a class cannot anticipate the class of objects it must create, or when a class wants its subclasses to specify the objects it creates.
- Example: The `java.util.Calendar` class has a `getInstance()` method that returns a Calendar object based on the current time zone and locale.

2. Prototype Design Pattern:

- Intent: The Prototype pattern involves creating new objects by copying an existing object, known as the prototype. This pattern is used to create a new object by copying an existing object, known as the prototype.
- Usage: When the cost of creating an object is more expensive or complex than copying an existing one, and when instances of a class have only a few different combinations of state.
- Example: Cloning in Java uses the Prototype pattern. The `clone()` method creates a new object by copying an existing object.

3. Builder Design Pattern:

- Intent: The Builder pattern is used to construct a complex object step by step. It separates the construction of a complex object from its representation so that the same construction process can create different representations.
- Usage: When an object needs to be constructed with numerous configuration options or when the construction process should be independent of the resulting object's representation.
- Example: The construction of an object in the StringBuilder class in Java, where you can append strings and other data types to build a final string.

The Factory pattern is concerned with creating objects without specifying their exact class.The Prototype pattern involves creating new objects by copying an existing object.The Builder pattern is used to construct a complex object step by step, allowing for different representations.

## 2 ) Give an example which is suitable for all three design pattern i.e. factory, builder and prototype design pattern:

Here we have a clothing store that produces different types of clothing items (Factory), allows customization of clothing items (Builder), and can create variations of existing clothing items.

- Code:

```
// Interface representing common properties of all clothing items
interface ClothingItem {
    void displayDetails();
}

// Concrete implementation of T-shirt
class TShirt implements ClothingItem {
    private String brand;
    private String size;
    private String color;
    private String material;

    // Constructor for T-shirt
    public TShirt(String brand, String size, String color, String
material) {
        this.brand = brand;
        this.size = size;
        this.color = color;
        this.material = material;
    }

    @Override
    public void displayDetails() {
        System.out.println("\nT-Shirt -\n Brand: " + brand + "\n Size:
" + size + "\n Color: " + color + "\n Material: " + material+"\n");
    }
}

// Builder interface for constructing clothing items
interface ClothingItemBuilder {
    void buildBrand(String brand);
```

```
    void buildSize(String size);

    void buildColor(String color);

    void buildMaterial(String material);

    ClothingItem getResult();
}

// Concrete implementation of TShirtBuilder
class TShirtBuilder implements ClothingItemBuilder {
    private String brand;
    private String size;
    private String color;
    private String material;

    @Override
    public void buildBrand(String brand) {
        this.brand = brand;
    }

    @Override
    public void buildSize(String size) {
        this.size = size;
    }

    @Override
    public void buildColor(String color) {
        this.color = color;
    }

    @Override
    public void buildMaterial(String material) {
        this.material = material;
    }

    @Override
    public ClothingItem getResult() {
        return new TShirt(brand, size, color, material);
    }
}
```

```
// Prototype interface for clothing items
interface ClothingItemPrototype {
    ClothingItemPrototype clone();

    void displayDetails();
}

// Concrete implementation of T-shirt prototype
class TShirtPrototype implements ClothingItemPrototype {
    private String brand;
    private String size;
    private String color;
    private String material;

    // Constructor for T-shirt prototype
    public TShirtPrototype(String brand, String size, String color,
String material) {
        this.brand = brand;
        this.size = size;
        this.color = color;
        this.material = material;
    }

    @Override
    public ClothingItemPrototype clone() {
        return new TShirtPrototype(brand, size, color, material);
    }

    @Override
    public void displayDetails() {
        System.out.println("\nT-Shirt Prototype -\n Brand: " + brand +
"\n Size: " + size + "\n Color: " + color + "\n Material: " +
material+"\n");
    }
}

// Factory for creating different types of clothing items
class ClothingItemFactory {
    public static ClothingItem createTShirt(String brand, String size,
String color, String material) {
        return new TShirt(brand, size, color, material);
    }
}
```

```
// Example usage of Factory, Builder, and Prototype patterns
public class Main {
    public static void main(String[] args) {
        // Factory pattern: Creating a T-shirt using the factory
        ClothingItem tShirtFromFactory =
ClothingItemFactory.createTShirt("Prada", "Large", "Blue", "Cotton");
        tShirtFromFactory.displayDetails();

        // Builder pattern: Creating a T-shirt using the builder
        ClothingItemBuilder tShirtBuilder = new TShirtBuilder();
        tShirtBuilder.buildBrand("Jack N Jones");
        tShirtBuilder.buildSize("Medium");
        tShirtBuilder.buildColor("Red");
        tShirtBuilder.buildMaterial("Polyester");
        ClothingItem tShirtFromBuilder = tShirtBuilder.getResult();
        tShirtFromBuilder.displayDetails();

        // Prototype pattern: Creating a T-shirt using the prototype
        ClothingItemPrototype tShirtPrototype = new
TShirtPrototype("MAX", "Small", "Green", "Silk");
        ClothingItemPrototype clonedTShirt = tShirtPrototype.clone();
        clonedTShirt.displayDetails();
    }
}
```

- Output:

```
T-Shirt -
 Brand: Prada
 Size: Large
 Color: Blue
 Material: Cotton


T-Shirt -
 Brand: Jack N Jones
 Size: Medium
 Color: Red
 Material: Polyester


T-Shirt Prototype -
 Brand: MAX
 Size: Small
 Color: Green
 Material: Silk
```

In this example:

- The ClothingItemFactory serves as a factory to create different types of clothing items. In this case, it creates T-shirts using the Factory pattern.
- The TShirtBuilder is a builder for constructing T-shirts with different details using the Builder pattern.
- The TShirtPrototype represents a prototype of a T-shirt, and the ClothingItemPrototype interface declares the clone method. We use the Prototype pattern to create a new T-shirt by cloning the prototype.

## 3 ) Give an example for which builder design pattern is more suitable than factory and prototype design pattern :

The Builder design pattern is typically more suitable than the Factory and Prototype design patterns when dealing with complex object creation scenarios, especially when you have a large number of optional parameters or configurations for an object.
Here we are designing a system for creating customized outfits with various optional features, such as size, color, fabric, and additional accessories. The Builder pattern can handle the complexity of creating custom outfits with different configurations.

- Code:

```
// Clothing class representing the complex object to be built
class Outfit {
    private String size;
    private String color;
    private String fabric;
    private boolean hasAccessories;

    // Private constructor to enforce the use of the builder
    private Outfit() {
        // Initialization logic if needed
    }

    // Getter methods for properties
    public String getSize() {
        return size;
    }
```

```
public String getColor() {
    return color;
}

public String getFabric() {
    return fabric;
}

public boolean hasAccessories() {
    return hasAccessories;
}

// Builder class for constructing Outfit objects
static class Builder {
    private Outfit outfit;

    Builder() {
        outfit = new Outfit();
    }

    Builder setSize(String size) {
        outfit.size = size;
        return this;
    }

    Builder setColor(String color) {
        outfit.color = color;
        return this;
    }

    Builder setFabric(String fabric) {
        outfit.fabric = fabric;
        return this;
    }

    Builder addAccessories() {
        outfit.hasAccessories = true;
        return this;
```

```
        }

        Outfit build() {
            return outfit;
        }
    }
}

// Example usage of the Builder pattern for creating outfits
public class Main {
    public static void main(String[] args) {
        // Create a custom outfit using the builder
        Outfit customOutfit = new Outfit.Builder()
                .setSize("Medium")
                .setColor("Blue")
                .setFabric("Cotton")
                .addAccessories()
                .build();

        // Use the created outfit
        System.out.println("Custom Outfit:");
        System.out.println("Size: " + customOutfit.getSize());
        System.out.println("Color: " + customOutfit.getColor());
        System.out.println("Fabric: " + customOutfit.getFabric());
        System.out.println("Accessories: " +
(customOutfit.hasAccessories() ? "Yes" : "No"));
    }
}
```

- Output :

```
Custom Outfit:
Size: Medium
Color: Blue
Fabric: Cotton
Accessories: Yes
```

## 4) Give an example with code for which factory design pattern is more suitable than builder and prototype design pattern :

- Code:

```
   interface ClothingItem {
    void displayDetails();
}


// Concrete implementation of T-shirt
class TShirt implements ClothingItem {
    private String brand;
    private String size;
    private String color;
    private String material;


    // Constructor for T-shirt
    public TShirt(String brand, String size, String color, String
material) {
        this.brand = brand;
        this.size = size;
        this.color = color;
        this.material = material;
    }


    @Override
    public void displayDetails() {
        System.out.println("\nT-Shirt- \n Brand: " + brand + "\n Size:
" + size + "\n Color: " + color + "\n Material: " + material);
    }
}


// Concrete implementation of Jeans
class Jeans implements ClothingItem {
    private String brand;
    private String size;
    private String color;
    private String material;
```

```
    // Constructor for Jeans
    public Jeans(String brand, String size, String color, String
material) {
        this.brand = brand;
        this.size = size;
        this.color = color;
        this.material = material;
    }


    @Override
    public void displayDetails() {
        System.out.println("\nJeans- \n Brand: " + brand + "\n Size: "
+ size + "\n Color: " + color + "\n Material: " + material + "\n");
    }
}


// ClothingFactory responsible for creating different types of clothing
items
class ClothingFactory {
    public static ClothingItem createTShirt(String brand, String size,
String color, String material) {
        return new TShirt(brand, size, color, material);
    }


    public static ClothingItem createJeans(String brand, String size,
String color, String material) {
        return new Jeans(brand, size, color, material);
    }
}


// Example usage of the Factory pattern for creating T-shirts and Jeans
public class Main {
    public static void main(String[] args) {
        // Create a T-shirt using the factory
        ClothingItem tShirt = ClothingFactory.createTShirt("Gucci",
"Large", "Blue", "Cotton");
        tShirt.displayDetails();
```

```
        // Create Jeans using the factory
        ClothingItem jeans = ClothingFactory.createJeans("Tommy
Hilfiger", "Medium", "Black", "Denim");
        jeans.displayDetails();
    }
}
```

- Output:

```
 T-Shirt-
  Brand: Gucci
  Size: Large
  Color: Blue
  Material: Cotton

 Jeans-
  Brand: Tommy Hilfiger
  Size: Medium
  Color: Black
  Material: Denim
```

## 5) Give an example with code for which prototype design pattern is more suitable than builder and factory design pattern :

```
interface ClothingPrototype {
    ClothingPrototype clone();
    void displayDetails();
}

// Concrete implementation of T-shirt prototype
class TShirtPrototype implements ClothingPrototype {
    private String brand;
    private String size;
    private String color;
    private String material;

    // Constructor for T-shirt prototype
    public TShirtPrototype(String brand, String size, String color,
String material) {
        this.brand = brand;
        this.size = size;
        this.color = color;
```

```
        this.material = material;
    }

    @Override
    public ClothingPrototype clone() {
        return new TShirtPrototype(brand, size, color, material);
    }

    @Override
    public void displayDetails() {
        System.out.println("\nT-Shirt- \n Brand: " + brand + "\n Size:
" + size + "\n Color: " + color + "\n Material: " + material + "\n");
    }
}

// Concrete implementation of Jeans prototype
class JeansPrototype implements ClothingPrototype {
    private String brand;
    private String size;
    private String color;
    private String material;

    // Constructor for Jeans prototype
    public JeansPrototype(String brand, String size, String color,
String material) {
        this.brand = brand;
        this.size = size;
        this.color = color;
        this.material = material;
    }

    @Override
    public ClothingPrototype clone() {
        return new JeansPrototype(brand, size, color, material);
    }

    @Override
    public void displayDetails() {
        System.out.println("\nJeans- \n Brand: " + brand + "\n Size: "
+ size + "\n Color: " + color + "\n Material: " + material+ "\n");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        // Create T-shirt prototype with different details
        ClothingPrototype tShirtPrototype = new TShirtPrototype("Vero
Moda", "Medium", "Red", "Polyester");

        // Clone T-shirt prototype to create a new instance
        ClothingPrototype newTShirt = tShirtPrototype.clone();
        newTShirt.displayDetails();

        // Create Jeans prototype with different details
        ClothingPrototype jeansPrototype = new JeansPrototype("Levi's",
"Large", "Blue", "Denim");

        // Clone Jeans prototype to create a new instance
        ClothingPrototype newJeans = jeansPrototype.clone();
        newJeans.displayDetails();
    }
}
```

- Output:

```
T-Shirt-
 Brand: Vero Moda
 Size: Medium
 Color: Red
 Material: Polyester


Jeans-
 Brand: Levi's
 Size: Large
 Color: Blue
 Material: Denim
```