```
/*
The following is an example of a Pet Factory
implementing the singleton design pattern
*/

public class PetFactory implements Factory {

  // Stores the instance of a pet factory.
  protected static PetFactory instance;

  // Protected constructor for pet factory.
  protected PetFactory(){}

  // Retrieves the instance for a pet factory
  public static PetFactory getInstance){
    if (this.instance == null){
        this.instance = new PetFactory();
    }
    return this.instance;
  }

  public Pet createPet(){
    // Creates pet
    // Complex
  }
}

public class MockPetFactory extends
PetFactory{
  public setInstance(PetFactory factory){
    this.instance = factory
  }

  public Pet createPet(){
    // Creates pet
    // Simplified
  }
}
```

Breaking Dependencies w/ Singleton

- There are two primary ways to work with breaking dependencies of a singleton.
  - Is the Singleton property necessary?
    - 🟩 No
      - Relax the singleton property by making the constructor public, creating a new instance, and allowing the setting of a new instance.
      - [Optionally] you can convert the class to a regular class.
    - 🟨 Yes
      - Permeate the class constructor and instance variable from private —> protected.
      - Create a child class of the singleton (used only for testing purposes).
      - Write a setter for the instance in the child class.
  - If there are a large number of global instance variables , then there may be deeper structural deficiencies in the code, such as failing to follow SRP.

```java
/*
The following is an example of a Pet Factory
implementing the singleton design pattern
*/

public class PetFactory implements Factory {

  // Stores the instance of a pet factory.
  protected static PetFactory instance;

  // Protected constructor for pet factory.
  protected PetFactory(){}

  // Retrieves the instance for a pet factory
  public static PetFactory getInstance){
    if (this.instance == null){
        this.instance = new PetFactory();
    }
    return this.instance;
  }

  public Pet createPet(){
    // Creates pet
    // Complex
  }
}

public class MockPetFactory extends
PetFactory{
  public setInstance(PetFactory factory){
    this.instance = factory
  }

  public Pet createPet(){
    // Creates pet
    // Simplified
  }
}
```

Demo Part 1