**The assignment: A flexible inventory system**

Your task is to build a basic inventory system for a store. The system should be able to manage different types of products (e.g., electronics, food) and perform generic operations on them.

**Step 1: Define the base Product class**

First, create a non-generic Product class. This will be the base class that all your specific product types will extend.

1. **Create an abstract class Product**.
2. This class should have:  
   \* A private String name.  
   \* A private double price.  
   \* A constructor to initialize the name and price.  
   \* A public String getName() getter.  
   \* A public double getPrice() getter.  
   \* An abstract public String getDetails() method. This will be implemented by specific product types to return a unique description.
3. **Override the toString() method** to provide a default string representation of the product, including its name and price.

**Guide:**

* Remember to use the abstract keyword for both the class and the getDetails() method.
* The toString() method should produce a string like "Product: [Name], Price: [Price]".

java

*// Example starter code for Product*

public abstract class Product {

private String name;

private double price;

public Product(String name, double price) {

this.name = name;

this.price = price;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

public abstract String getDetails();

@Override

public String toString() {

return "Product: " + name + ", Price: $" + price;

}

}

Use code with caution.

**Step 2: Create specific product classes**

Next, create a few concrete classes that extend Product.

1. **Create a class Electronics extends Product**.
   * Add a private String manufacturer field.
   * Implement the constructor to call the superclass constructor and initialize the new field.
   * Override the getDetails() method to return a string like "Manufacturer: [Manufacturer]".
2. **Create a class Food extends Product**.
   * Add a private String expirationDate field.
   * Implement the constructor.
   * Override the getDetails() method to return a string like "Expires: [ExpirationDate]".

**Guide:**

* Use the super() keyword in your constructors to call the Product constructor.

**Step 3: Build the generic Inventory class**

Now, create the core of your system, a generic Inventory class that can hold any type of Product. This is where you will use your knowledge of generics, collections (List), and bounded type parameters.

1. **Define a generic class Inventory<T extends Product>**. This uses a bounded type parameter to ensure the inventory can only hold Products or their subclasses.
2. **Use a private List<T> items = new ArrayList<>();** to store the products.
3. **Add the following methods:**
   * public void addProduct(T item): Adds a product to the inventory.
   * public T getProduct(int index): Retrieves a product from the inventory.
   * public void printInventory(): Prints a detailed list of all products in the inventory, including their specific details. This should call the getDetails() method of each item.

**Guide:**

* Remember that because T is bounded by Product, you can safely call methods from the Product class on any object of type T.
* You will iterate through the items list and print each one.

**Step 4: Implement generic utility methods**

Create a separate utility class with static generic methods to demonstrate working with different types of products.

1. **Create a class InventoryUtils**.
2. **Create a generic method public static <T extends Product> double calculateTotalPrice(List<T> products)**. This method should:
   * Accept a List of any type T that extends Product.
   * Return the sum of the prices of all products in the list.
3. **Create a generic method public static <T> void printItems(List<T> items)**. This method should:
   * Accept a List of any type T.
   * Print each item in the list. This demonstrates the basic use of a generic method that is not bounded.

**Guide:**

* Notice the placement of the <T extends Product> for the bounded generic method.
* In the unbounded generic method printItems(), T can be any object, so you will just print the item directly, relying on its toString() method.

**Step 5: Test your code in the Main class**

Finally, put it all together in a public class Main.

1. **Create an Inventory<Electronics>** and add a few Electronics products.
2. **Create an Inventory<Food>** and add a few Food products.
3. **Print the inventory for both** to show that they function correctly.
4. **Try to add a Food object to your Inventory<Electronics>**. The compiler should prevent this with a type error. **(Don't uncomment this code; just observe the error).**
5. **Use the InventoryUtils.calculateTotalPrice() method** to find the total value of your electronics and food inventories.
6. **Use the InventoryUtils.printItems() method** to print both lists of products.

**Guide:**

* Remember that List<T> is needed for the generic methods in InventoryUtils. To get the list of products from your Inventory class, you might need to add a getList() method to your Inventory class or create a new list for testing purposes.

**My guidance: Checkpoints and hints**

**Checkpoint 1: Generic Classes**

* **Your task:** Implement Product, Electronics, Food, and Inventory.
* **Hint:** Pay close attention to the extends keyword in the Inventory class declaration: class Inventory<T extends Product>. This is crucial for bounded types.

**Checkpoint 2: Generic Methods**

* **Your task:** Implement InventoryUtils and its two static generic methods.
* **Hint:** Remember that the type parameter <T> for a generic method goes *before* the return type, like public static <T> void.... The bounded version is <T extends Product>.

**Checkpoint 3: Demonstration**

* **Your task:** Write the main method to prove everything works as expected.
* **Hint:** When you create Inventory instances, use concrete types: new Inventory<Electronics>() and new Inventory<Food>().
* To test the type safety, just write the incorrect line of code but comment it out with a note explaining why it won't compile. This demonstrates your understanding of the concept.

This assignment will force you to use all the major generics features in a practical and meaningful context. Good luck!