

Answer 1:

Product Class Hierarchy:

```
abstract class Product {  
    protected String name;  
    protected double price;  
    public Product(String name, double price) {  
        if (price <= 0) {  
            throw new IllegalArgumentException("Price must be positive.");  
        }  
        this.name = name;  
        this.price = price;  
    }  
    public double calculateTotalPrice(int quantity) {  
        if (quantity <= 0) {  
            throw new IllegalArgumentException("Quantity must be greater than zero.");  
        }  
        return price * quantity;  
    }  
    public String getName() {  
        return name;  
    }  
    public abstract String getDetails();  
}  
  
class Electronics extends Product {  
    private int warrantyMonths;  
    public Electronics(String name, double price, int warrantyMonths) {  
        super(name, price);  
        this.warrantyMonths = warrantyMonths;  
    }  
}
```

```

@Override

public String getDetails() {

    return "Electronics: " + name +

        ", Warranty: " + warrantyMonths + " months";

}

}

class Clothing extends Product {

    private String size;

    private String fabric;

    public Clothing(String name, double price, String size, String fabric) {

        super(name, price);

        this.size = size;

        this.fabric = fabric;

    }

    @Override

    public String getDetails() {

        return "Clothing: " + name +

            ", Size: " + size +

            ", Fabric: " + fabric;

    }

}

class Books extends Product {

    private String author;

    public Books(String name, double price, String author) {

        super(name, price);

        this.author = author;

    }

    @Override

    public String getDetails() {

```

```

        return "Book: " + name +
            ", Author: " + author;
    }
}

```

ShoppingCart Class:

```

import java.util.*;

class ShoppingCart {

    private Map<Product, Integer> cartItems = new HashMap<>();

    public void addProduct(Product product, int quantity) {

        if (product == null) {

            throw new IllegalArgumentException("Product cannot be null.");

        }

        if (quantity <= 0) {

            throw new IllegalArgumentException("Quantity must be greater than zero.");

        }

        cartItems.put(product, cartItems.getOrDefault(product, 0) + quantity);

    }

    public void viewCart() {

        if (cartItems.isEmpty()) {

            System.out.println("Cart is empty.");

            return;

        }

        double total = 0;

        for (Map.Entry<Product, Integer> entry : cartItems.entrySet()) {

            Product product = entry.getKey();

            int quantity = entry.getValue();

            double productTotal = product.calculateTotalPrice(quantity);

```

```

        System.out.println(product.getDetails() +
            ", Quantity: " + quantity +
            ", Total: $" + productTotal);
        total += productTotal;
    }

    System.out.println("Total Cart Price: $" + total);
}

public double calculateGrandTotal() {
    double total = 0;
    for (Map.Entry<Product, Integer> entry : cartItems.entrySet()) {
        total += entry.getKey()
            .calculateTotalPrice(entry.getValue());
    }
    return total;
}

public void clearCart() {
    cartItems.clear();
}
}

```

User Class:

```

class User {
    private String username;
    private String email;
    private ShoppingCart cart;
    public User(String username, String email) {
        this.username = username;
        this.email = email;
        this.cart = new ShoppingCart();
    }
}

```

```
}  
  
public ShoppingCart getCart() {  
    return cart;  
}  
  
public void viewCart() {  
    cart.viewCart();  
}  
  
public void finalizeCart() {  
    double total = cart.calculateGrandTotal();  
    if (total == 0) {  
        System.out.println("Your cart is empty. Add products first.");  
    } else {  
        System.out.println("Order placed successfully!");  
        System.out.println("Total amount paid: $" + total);  
        cart.clearCart();  
    }  
}  
}
```