GenericOrder<T>

Constructor: public GenericOrder()

Input: new GenericOrder()

Expected Output: Cannot instantiate the object as GenericOrder is an abstract class. So, the methods cannot be called.

ComputerOrder (extends GenericOrder<IComputerOrder>)

Constructor: public ComputerOrder()

Input: new ComputerOrder()

Expected Output: ComputerOrder object with empty ArrayList orders and random order number(UUID)

Getting Order Number: UUID getOrderId()

Input: new ComputerOrder().getOrderId()

Expected Output: Random UUID that generated when the ComputerOrder object created.

Add Item(Product) to the Order: void addItem(IComputerOrder item)

Input: Motherboard motherboard = new Motherboard(“manufacture”, 105.0f);

new ComputerOrder().addItem(motherboard)

Expected Output: motherboard object is added to the items ArrayList<IComputerPartyOrder>

Add inapposite Item to the Order: void addItem(IComputerOrder item)

Input: new ComputerOrder().addItem(new Cheddar(12.0f))

Expected Output: Cheddar object is not accepted as a parameter as the Cheddar(Cheese) class does not implement the ICompuerOrder interface. Compile Error.

Get Items in the Order: ArrayList<IComputerOrder> getItems()

Input: Motherboard motherboard = new Motherboard(“manufacture”, 105.0f);

new ComputerOrder().addItem(motherboard).getItems()

Expected Output: Items that added to the order will be returned. Arraylist with “motherboard” object will be returned in this case.

PartyTrayOrder (extends GenericOrder<IPartyTrayOrder>)

Constructor: public PartyTrayOrder()

Input: new PartyTrayOrder()

Expected Output: PartyTrayOrder object with empty ArrayList orders and random order number(UUID)

Getting Order Number: UUID getOrderId()

Input: new PartyTrayOrder().getOrderId()

Expected Output: Random UUID that generated when the PartyTrayOrder object created.

Add Correct Item(Product) to the Order: void addItem(IPartyTrayOrder item)

Input: new PartyTrayOrder().addItem(new Cheddar(12.0f));

Expected Output: Cheddar object is added to the items ArrayList<T>

Add inapposite Item to the Order: void addItem(IPartyTrayOrder item)

Input: new PartyTrayOrder().addItem(new Monitor(“mode”, 120.0f));

Expected Output: Monitor object is not accepted as a parameter as the Monitor(Cheese) class does not implement the IPartyTrayOrder interface. Compile Error.

Get Items(Product) in the Order: ArrayList<IPartyTrayOrder> getItems()

Input: new PartyTrayOrder().addItem(new Cheddar(12.0f)).getItems()

Expected Output: Expected Output: Items that added to the order will be returned. Arraylist with Cheddar object will be returned in this case.

OrderProcessor

Constructor: public OrderProcessor()

Input: new OrderProcessor()

Expected Output: OrderProcessor object will be created with its attributes.

Accepting Order: <T extends GenericOrder<?>> void accept(T order)

Input: ComputerOrder computerOrder = new ComputerOrder();

computerOrder.addItem(new Motherboard(“manufacture”, 105.0f));

OrderProcessor orderProcessor = new OrderProcessor();

orderProcessor. accept(new ComputerOrder());

Expected Output: ComputerOrder object will be added(accepted) to the OrderProcessor object’s orders arraylist.

Processing Order: void process()

Input: ComputerOrder computerOrder = new ComputerOrder();

computerOrder.addItem(new Motherboard(“manufacture”, 105.0f));

OrderProcessor orderProcessor = new OrderProcessor();

orderProcessor. accept(new ComputerOrder());

orderProcessor.process();

Expected Output: The accepted orders in the OrderProcessor object will be sorted by the product type(Computer Part, Peripheral, Cheese, Fruit, Service) and group by the order ID, then the orders in the “orders” arraylist will be removed as they processed.

Dispatching Order: void dispatchComputerParts()

Input: ComputerOrder computerOrder = new ComputerOrder();

computerOrder.addItem(new Motherboard(“manufacture”, 105.0f));

OrderProcessor orderProcessor = new OrderProcessor();

orderProcessor.accept(computerOrder);

orderProcessor.process();

orderProcessor.dispatchComputerParts();

Expected Output: Prints out accepted computer parts products and its order id and removes them from the processed list as they dispatched.

E.g.) ----------Computer Parts----------

Motherboard name=manufacture, price=105.0, order number=3675aee0-1163-466a-a1cb-9e851dacae13

Dispatching Order: void dispatchPeripherals()

Input: ComputerOrder computerOrder = new ComputerOrder();

computerOrder.addItem(new Printer("model", 105.0f));

OrderProcessor orderProcessor = new OrderProcessor();

orderProcessor.accept(computerOrder);

orderProcessor.process();

orderProcessor.dispatchPeripherals();

Expected Output: Prints out accepted Peripherals products and its order id and removes them from the processed list as they dispatched.

E.g.) ----------Peripherals----------

Printer model=model, price=105.0, order number=3675aee0-1163-466a-a1cb-9e851dacae13

Dispatching Order: void dispatchServices()

Input: ComputerOrder computerOrder = new ComputerOrder();

computerOrder.addItem(new AssemblyService("provider", 50.0f));

OrderProcessor orderProcessor = new OrderProcessor();

orderProcessor.accept(computerOrder);

orderProcessor.process();

orderProcessor.dispatchServices();

Expected Output: Prints out accepted Services products and its order id and removes them from the processed list as they dispatched.

E.g.) ----------Services----------

AssemblyService provider=provider, price=50.0, order number=3675aee0-1163-466a-a1cb-9e851dacae13

Dispatching Order: void dispatchCheese()

Input: PartyTrayOrder partyTrayOrder= new PartyTrayOrder();

partyTrayOrder.addItem(new Cheddar(10.0f));

OrderProcessor orderProcessor = new OrderProcessor();

orderProcessor.accept(partyTrayOrder);

orderProcessor.process();

orderProcessor.dispatchCheese();

Expected Output: Prints out accepted Cheeses products and its order id and removes them from the processed list as they dispatched.

E.g.) ----------Cheeses----------

Cheddar price=10.0, order number=3675aee0-1163-466a-a1cb-9e851dacae13

Dispatching Order: void dispatchFruit()

Input: PartyTrayOrder partyTrayOrder= new PartyTrayOrder();

partyTrayOrder.addItem(new Apple(10.0f));

OrderProcessor orderProcessor = new OrderProcessor();

orderProcessor.accept(partyTrayOrder);

orderProcessor.process();

orderProcessor.dispatchFruit ();

Expected Output: Prints out accepted Fruits products and its order id and removes them from the processed list as they dispatched.

E.g.) ----------Fruits----------

Apple price=0.0, order number=3675aee0-1163-466a-a1cb-9e851dacae13

MotherboardGenerator

RAMGenerator

DriveGenerator

PeripheralGenerator

DeliveryServiceGenerator

AssemblyServiceGenerator

CheeseFruitGenerator

Above Generators created for the testing purposes to generate random data, based on TIJ.

Constructor: *****Generator(Class<T> type)

Input: E.g.) new CheeseFruitGenerator(Apple.class)

Expected Output: Random data generator object is created for Apple.class

Get data: T next()

Input: E.g.) new CheeseFruitGenerator(Apple.class).next()

Expected Output: return random Apple object using the preset attributes

Create Generator: <T> Generator<T> create(Class<T> type)

Input: E.g.) CheeseFruitGenerator.create(Apple.class)

Expected Output: return CheeFruitGenerator object. Just a helper function