1.) Prepaid Shop - Mobile Service Provider

Create a simple HTML form where users are able to charge their mobile phones with a given amount of money.

```
Possible amounts: $10, $30, $50, $100
The 3 mobile service providers: "Provider A", "Provider B", "Provider C"
Mobile phone number: 7 numbers.
```

Create a servlet which gets the HTTP GET request from the HTML form and checks the validity of the messages. If the message is invalid, then returns an “Invalid data” text, else writes “Ok”.

- Create an inner class within servlet and store these payments in a java ArrayList.
- Create an other method to list the number of payments (with a new HTTP get request).

2.) Message queue

flowchart LR subgraph Client1 [Client 1: Color Generator] direction TB start1([Start]) -->|Every 2s| sendColors{Send Colors} sendColors -->|RED, GREEN, BLUE| ColorQueue([colorQueue]) end subgraph Processor [Message Processor] direction TB ColorQueue --> receiveColors[Receive Colors] receiveColors -->|Count messages per color| checkCount{Check Count} checkCount -->|10 messages of same color?| sendStat checkCount -->|Less than 10| receiveColors sendStat[Send '10 color processed'] --> ColorStatsQueue([colorStatistics Queue]) end subgraph Client3 [Client 3: Statistics Reporter] direction TB ColorStatsQueue --> readStats[Read Statistics] readStats --> displayConsole[Display on Console] end classDef startend fill:#f9f,stroke:#333,stroke-width:2px; class start1 startend; classDef operation fill:#bbf,stroke:#fff,stroke-width:2px; class sendColors,receiveColors,displayConsole operation; classDef decision fill:#fbf,stroke:#333,stroke-width:2px; class checkCount decision; classDef process fill:#ccf,stroke:#333,stroke-width:2px; class readStats,sendStat process;
Create an application consisting of three clients. The first client connects to the 'colorQueue' message queue using point-to-point connection and sends messages with randomly assigned parameters (RED, GREEN, and BLUE), every 2 seconds. Create a message processor that receive messages with the 'RED', 'GREEN', and 'BLUE' parameters exclusively. After receiving every 10 messages of the same color, the processor sends a message to the 'colorStatistics' queue indicating that they have processed 10 messages of a given color.

Create a third client that reads the statistics from the 'colorStatistics' queue and outputs to the console, for example, '10 'RED' messages have been processed'.

Create tests to validate the original functionality

3.) Temperature Alert System

Create an application that simulates temperature monitoring using message queues. This system involves three separate clients for generating, processing, and reporting temperature data.

Component 1: Temperature Generation Client

- Client 1 connects to the **temperatureQueue** using a point-to-point connection.
- Function: Sends messages with randomly assigned temperature readings (in degrees Celsius) within a normal range (-10 to 40 degrees Celsius) every 3 seconds.

Component 2: Temperature Monitoring Processor

- Message Processor: Exclusively receives temperature readings from temperatureQueue.
- Function: Monitors for abnormal temperature readings. A reading is considered abnormal if it is below -5 or above 35 degrees Celsius.
- Notification: After receiving 5 abnormal readings, the processor sends a notification to the **alertQueue**, indicating that 5 abnormal temperature readings have been detected.

Component 3: Alert Reporting Client

- Reads from the **alertQueue**.
- Output: Prints a message to the console, e.g., “5 abnormal temperature readings have been detected.”

Tests to validate:

- The functionality of sending and receiving messages correctly.
- The correct identification of abnormal temperature readings.
- The accurate accumulation and dispatch of alerts after every 5 abnormal readings.

4.) Humidity Control System

Create an application that simulates monitoring and controlling humidity levels within an environment using message queues. This system will involve three separate clients for generating, processing, and reporting humidity data.

Component 1: Humidity Generation Client

- Client 1: Connects to the **humidityQueue** using a point-to-point connection.
- Function: Sends messages with randomly assigned humidity readings (expressed as percentages) within a range of 30% to 90% every 4 seconds.

Component 2: Humidity Monitoring Processor

- Message Processor: Receives humidity readings from **humidityQueue**.
- Function: Monitors for high humidity readings, considered high if they exceed 70%.
- Notification: After receiving 3 consecutive high humidity readings, the processor sends a notification to the **alertQueue**, indicating that high humidity levels have been detected.

Component 3: Alert Reporting Client

- Client 3: Reads from the **alertQueue**.
- Output: Prints a message to the console, e.g., “High humidity alert: 3 consecutive readings over 70%.”

Create tests to validate:

- The functionality of sending and receiving messages correctly.

- The correct identification of high humidity readings.
- The accurate accumulation and dispatch of alerts after detecting high humidity conditions.

5.) Air Quality Monitoring System

Create an application that simulates monitoring and controlling air quality levels within an indoor environment using message queues. This system will involve three separate clients for generating, processing, and reporting air quality data.

Component 1: Air Quality Data Generation Client

- Client 1: Connects to the **airQualityQueue** using a point-to-point connection.
- Function: Sends messages with randomly assigned air quality readings, quantified as Air Quality Index (AQI) values, within a range of 0 (Good) to 300 (Hazardous) every 3 seconds.

Component 2: Air Quality Alert Processor Message Processor: Exclusively receives AQI readings from **airQualityQueue**. Function: Monitors for poor air quality readings, considered poor if they exceed an AQI of 150. Notification: After receiving 2 consecutive readings above the threshold, the processor sends a notification to the **airQualityAlertQueue**, indicating that poor air quality levels have been detected.

Component 3: Alert Reporting Client

- Client 3: Reads from the **airQualityAlertQueue**.
- Output: Prints a message to the console, e.g., “Air quality alert: 2 consecutive readings above 150 AQI.”

Create tests to validate:

- The functionality of sending and receiving messages correctly.
- The correct identification of poor air quality readings.
- The accurate accumulation and dispatch of alerts after detecting poor air quality conditions.