Java Problem Set Worksheet 1.J CMPE 261

Osman Emre Tutay

December 2025

Problem 1: Java Extending Thread Class

Design a Java class named `RocketLaunch` to simulate a countdown timer for space missions.

- **RocketLaunch Class:** Extends the `Thread` class.
  - **Attributes:** `missionName` (String).
  - **Constructor:** Initializes the mission name.
- **Methods:**
  - Override `run()`: Executes a countdown from 5 to 1. Prints "`[missionName]: T-minus [seconds]`" with a 1-second pause (`Thread.sleep(1000)`) between counts. After reaching 0, print "`[missionName]: LIFTOFF!`".

Instructions:

- Create two instances of `RocketLaunch` (e.g., "Apollo" and "Voyager") in the main method.
- Call `start()` on both objects to execute them concurrently.
- *Note:* Handle the `InterruptedException` inside the `run` method.

Sample Output:

```
Apollo: T-minus 5
Voyager: T-minus 5
Apollo: T-minus 4
Voyager: T-minus 4
...
Apollo: LIFTOFF!
Voyager: LIFTOFF!
```

Problem 2: Java Implementing Runnable Interface

Create a class named `DataBackup` to demonstrate defining tasks separate from the thread execution mechanism.

- **DataBackup Class:** Implements the `Runnable` interface.
  - **Attributes:** `dbName` (String).
  - **Constructor:** Initializes the database name.
- **Methods:**
  - Override `run()`: Prints "Backing up [dbName]...", simulates a delay, and prints "[dbName] backup complete."

Instructions:

- Create a `DataBackup` object for "UserDB".
- Create a `Thread` object, passing the `DataBackup` instance to its constructor.
- Start the thread.
- Print "Main thread continuing other tasks..." immediately after starting the backup thread to demonstrate concurrency.

Sample Output:

```
Main thread continuing other tasks...
Backing up UserDB...
UserDB backup complete.
```

Problem 3: Java Thread Methods (Join & Sleep)

Simulate a relay race using the `join()` method to ensure one runner cannot start until the previous one finishes.

- **RelayRunner Class:** Extends `Thread`.
  - **Attributes:** `runnerName` (String) and `speed` (int, milliseconds to sleep).
- **Methods:**
  - Override `run()`: Prints "[runnerName] took the baton.", sleeps for `speed` milliseconds, and prints "[runnerName] passed the baton."

Instructions:

- Create three `RelayRunner` objects (e.g., "Runner 1", "Runner 2", "Runner 3").
- Start "Runner 1".
- Use `join()` on "Runner 1" to force the main thread to wait until it finishes before starting "Runner 2".
- Repeat the logic so "Runner 3" only starts after "Runner 2" finishes.

Sample Output:

```
Runner 1 took the baton.
Runner 1 passed the baton.
Runner 2 took the baton.
Runner 2 passed the baton.
Runner 3 took the baton.
Runner 3 passed the baton.
Race Finished!
```

Problem 4: Java Thread Methods (Interrupt & isAlive)

Create a security system monitor that runs indefinitely until an admin stops it.

- **SecuritySensor Class:** Extends `Thread`.
- **Methods:**
  - Override `run()`: Use a `while` loop that continues as long as `!Thread.interrupted()`. Inside the loop, print "Sensor Active: Scanning area..." and sleep for 1000ms.
  - Inside the catch block for `InterruptedException`, print "Sensor Alert: System shutting down manually."

Instructions:

- Instantiate and start the `SecuritySensor`.
- In the main method, sleep for 3000ms to let the sensor run a few times.
- Check if the thread is running using `isAlive()` and print "System status: [true/false]".
- Call `interrupt()` on the sensor thread to stop it.

Sample Output:

```
Sensor Active: Scanning area...
Sensor Active: Scanning area...
Sensor Active: Scanning area...
System status: true
Sensor Alert: System shutting down manually.
```

Problem 5: Java Thread Priority

Demonstrate the effect of thread priorities using a `DownloadTask` class.

- **DownloadTask Class:** Extends `Thread`.
  - **Attributes:** `fileName` (String).
  - **Methods:**
    - Override `run()`: Prints "Downloading [fileName]..."

Instructions:

- Create two `DownloadTask` objects: `criticalPatch` ("System Security Patch") and `backgroundUpdate` ("Wallpaper Pack").
- Set the priority of `criticalPatch` to `Thread.MAX_PRIORITY`.
- Set the priority of `backgroundUpdate` to `Thread.MIN_PRIORITY`.
- Start both threads.

Sample Output:

```
Downloading System Security Patch...
Downloading Wallpaper Pack...
```