1. **Threadsynchronisation - Stack**

   Schreiben Sie eine Klasse `SyncronizedStack`. Diese Klasse verwaltet in einem Feld Referenzen auf Objekte eines generischen Datentyps `<E>` nach dem LIFO (Last In, First Out) Prinzip:

```
public class SynchronizedStack<E> {
   private E []elements;     // Datenfeld
   private int size;         // aktuelle Stackgröße

   @SuppressWarnings("unchecked")
   public Stack(int capacity) {
     s=(E[]) new Object[capacity];
   }

   public void push(E e)    // legt ein neues Objekt auf den Stack
   { /* ... */ }
   public E pop()           // entfernt und liefert das oberste Objekt
   { /* ... */ }

   // weitere notwendige Methoden
}
```

   Gegeben ist außerdem eine zugehörige Testklasse `SynchronizedStackTest`.

   a) Stellen Sie die Klasse `Stack` testgetrieben fertig. Die Klasse soll dabei threadsicher Elemente auf den Stack legen und von diesem entfernen können.

   b) Schreiben Sie ein Javaprogramm, in dem 2 Threads (ein Produzent und ein Konsument) konkurrierend auf ein Stackobjekt zugreifen. Die beiden Threads sollen in gleichverteilten zufälligen Intervallen Daten auf den Stack schreiben und vom Stack lesen.
   Experimentieren Sie mit dem Programm, indem Sie den Produzenten rascher produzieren lassen als der Konsument liest und umgekehrt. Protokollieren Sie auf der Kommandozeile alle Schreib- und Lesevorgänge mit und geben Sie nach jeder Änderung den Zustand des Stacks aus.

2. **Lagersimulation**

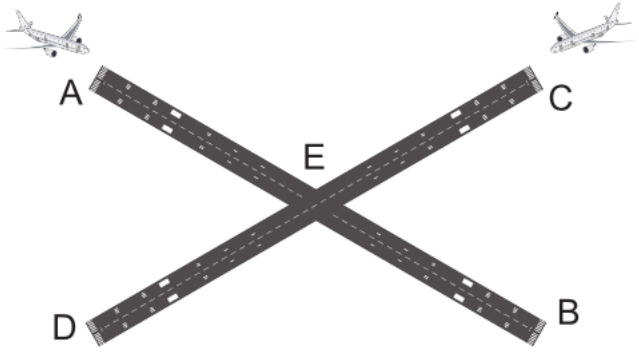
   Entwickeln Sie eine threadbasierte Simulation zu folgendem Szenario. Verwenden Sie zur Synchronisation Klassen aus dem Paket `java.util.concurrent`.

   - Ein Warenlager hat eine Lagerkapazität von 20 Kubikmetern.
   - Ein Produzententhread liefert in gleichverteilten Zeitintervallen von 1 bis 5 Sekunden Waren mit einem zufälligen Raumbedarf von 5 bis 10 Kubikmetern. Lassen Sie den Produzententhread in einer Schleife 35mal Waren liefern.
   - Ein Konsumententhread holt die Waren aus dem Lager ab. In gleichverteilten Zeitintervallen von 3 bis 6 Sekunden werden jeweils genau 7 Kubikmeter abgeholt. Am Ende der Simulation soll das Lager ausgeräut werden, bei der letzten Abholung dürfen also auch weniger als 7 Kubikmeter abgeholt werden.

- Das Lager darf nur von einem der Threads gleichzeitig benutzt werden. Stellen Sie die Einhaltung dieser Bedingung durch Threadsynchronisation sicher.
- Verwenden Sie für die Kommunikation der beiden Threads die Methoden `await()` und `signal()`. Der Produzent muss warten, wenn der verfügbare Raum im Lager kleiner ist als der Platzbedarf der aktuellen Lieferung. Der Konsument muss warten, wenn weniger als 7 Kubikmeter Ware im Lager sind und der Produzent noch arbeitet.
- Protokollieren Sie jede erfolgreiche Lieferung und jede erfolgreiche Abholung inklusive Timestamp und neuem Lagerstand sowohl auf der Standardausgabe als auch ein eine Textdatei `lager.log`.
- Protokollieren Sie außerdem auf der Standardausgabe inklusive Timestamp, wenn der Produzent wegen zu vollem Lager oder der Konsument wegen zu leerem Lager warten muss.
- Schreiben Sie auch eine FX-Anwendung, welche das Lager und die Tätigkeiten von Produzent und Konsument visualisiert.

3. Flugpatzsimulation

Ein Flugplatz hat zwei einander im Punkt E kreuzende Landebahnen AB und CD (vgl. Abbildung). Ankommende Flugzeuge wählen entweder die Landebahn AB oder CD für die Landung.



Dabei gelten die folgenden Regeln:

(a) Jede Landebahn darf zu einem bestimmten Zeitpunkt nur von einem Flugzeug verwendet werden.

(b) Befindet sich ein Flugzeug auf einer Landebahn vor dem Punkt E, so sperrt es auch die andere Landebahn. Befindet es sich nach dem Punkt E, so ist die andere Landebahn frei.

Erstellen Sie zu diesem Szenario eine threadbasierte Simulation für den Landvorgang von 20 Flugzeugen nach folgenden Vorgaben:

- Modellieren Sie die Flugzeuge als Runnable - Instanzen. Jedem Flugzeug wird bei der Instanziierung mitgeteilt, ob es auf Landebahn AB oder CD landen soll. Jedes Flugzeug belegt genau 3 Sekunden die Landebahn vor dem Punkt E und 5 Sekunden nach dem Punkt E.
- Sorgen Sie mit ReentrantLocks und Conditions dafür, dass die oben festgelegten Regeln eingehalten werden.

- Schreiben Sie eine Konsoleapplikation, in der Sie in zufällig gleichverteilten Zeitintervallen von 0 - 10 Sekunden 20 Flugzeuge erzeugen und führen Sie die zugehörigen Runnable-Instanzen in einem ThreadPool der fixen Größe 20 aus. Protokollieren sich auf der Konsole mit, wenn ein Flugzeug eintrifft, auf die Benutzung der Landebahn wartet, aufsetzt, den Punkt E passiert bzw. die Landebahn wieder verlässt. Ihr Protokoll könnte etwa wie folgt aussehen:

```
Plane 1 arrives for runway AB
Plane 1 is on runway AB
Plane 1 crosses E on runway AB
Plane 1 leaves runway AB
Plane 2 arrives for runway AB
Plane 2 is on runway AB
Plane 2 crosses E on runway AB
Plane 3 arrives for runway CD
Plane 2 leaves runway AB
Plane 3 is on runway CD
Plane 3 crosses E on runway CD
Plane 4 arrives for runway CD
Plane 5 arrives for runway CD
Plane 6 arrives for runway CD
Plane 3 leaves runway CD
Plane 4 is on runway CD
Plane 7 arrives for runway CD
Plane 4 crosses E on runway CD
Plane 4 leaves runway CD
...
```