Exercice 1

1. **Rôles des concepts en programmation Java :**
    - **GUI (Graphical User Interface)** : Permet de créer des interfaces utilisateur interactives. Utilisée pour améliorer l'expérience utilisateur en offrant des composants visuels (boutons, menus, fenêtres, etc.).
    - **JDBC (Java Database Connectivity)** : Fournit une API pour accéder aux bases de données relationnelles à partir de Java. Utilisée pour exécuter des requêtes SQL et gérer les connexions à la base de données.
    - **File I/O (Input/Output)** : Gère la lecture et l'écriture de données vers et depuis des fichiers. Utilisé pour stocker et récupérer des informations sur le disque.
    - **Applet** : Programme Java qui s'exécute dans un navigateur web. Utilisé pour ajouter des fonctionnalités dynamiques et interactives aux pages web.
    - **Asynchronous** : Permet d'exécuter des opérations de manière non bloquante. Utilisé pour améliorer la performance en permettant à d'autres tâches de s'exécuter simultanément.
2. **Éléments importants du cycle de vie, rôle et syntaxe d'utilisation :**
    - **Une classe Java** :
        - **Cycle de vie** : Déclaration, instanciation, utilisation, destruction.
        - **Rôle** : Modéliser des objets avec des attributs et des comportements.
        - **Syntaxe** :

```
java
Copier le code
public class MyClass {
    private int myField;
    public MyClass(int field) {
        this.myField = field;
    }
    public void myMethod() {
        // Code here
    }
}
```

    - **Un programme graphique** (JavaFX) :
        - **Cycle de vie** : Initialisation, start, exécution, arrêt.
        - **Rôle** : Créer des applications GUI riches.
        - **Syntaxe** :

```
java
Copier le code
import javafx.application.Application;
import javafx.stage.Stage;

public class MyApp extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Setup GUI here
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

- **Connexion à une base de données MySQL** :
  - **Cycle de vie** : Chargement du driver, création de la connexion, utilisation, fermeture.
  - **Rôle** : Interagir avec une base de données MySQL.
  - **Syntaxe** :

```
java
Copier le code
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class MySQLConnection {
    public static void main(String[] args) {
        try {
            Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/
mydb", "user", "password");
            // Use the connection
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- **Traitement d'un fichier** :
  - **Cycle de vie** : Ouverture, lecture/écriture, fermeture.
  - **Rôle** : Lire ou écrire des données vers/depuis des fichiers.
  - **Syntaxe** :

```
java
Copier le code
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderExample {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new
FileReader("file.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- **Traitement des erreurs** :
  - **Cycle de vie** : Détection, propagation, gestion.
  - **Rôle** : Gérer les situations exceptionnelles et assurer la robustesse.
  - **Syntaxe** :

```
java
```

```
Copier le code
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero");
        }
    }
}
```

3. **Cycle de vie d'une Applet Java et rôle de chacune de ses méthodes :**
   - **Cycle de vie** : init(), start(), paint(), stop(), destroy().
   - **init()** : Initialisation de l'applet.
   - **start()** : Démarrage de l'applet.
   - **paint()** : Dessin de l'applet.
   - **stop()** : Arrêt de l'applet.
   - **destroy()** : Destruction de l'applet.

Exercice 2

4. **Daemon thread en Java :**
   - Un daemon thread est un thread de bas niveau utilisé pour des tâches de fond. Sa principale caractéristique est qu'il ne bloque pas la terminaison de l'application; lorsqu'il n'y a plus de threads non-daemon en cours d'exécution, la JVM s'arrête, tuant tous les threads daemon restants.
5. **Programme Java avec un daemon thread et CompletableFuture :**

```
java
Copier le code
public class DaemonExample {
    public static void main(String[] args) {
        Thread daemonThread = new Thread(() -> {
            try {
                while (true) {
                    System.out.println("Message every second");
                    Thread.sleep(1000);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        daemonThread.setDaemon(true);
        daemonThread.start();

        CompletableFuture.runAsync(() -> {
            try {
                Thread.sleep(5000); // Simulating long operation
                System.out.println("Long operation completed");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        try {
            Thread.sleep(7000);
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

6. **Définir la priorité d'un thread en Java et effet sur l'ordonnancement :**

```
java
Copier le code
public class ThreadPriorityExample {
    public static void main(String[] args) {
        Thread highPriority = new Thread(() ->
System.out.println("High priority thread"));
        Thread lowPriority = new Thread(() -> System.out.println("Low
priority thread"));

        highPriority.setPriority(Thread.MAX_PRIORITY);
        lowPriority.setPriority(Thread.MIN_PRIORITY);

        lowPriority.start();
        highPriority.start();
    }
}
```

   - **Effet de la priorité** : La priorité des threads influence l'ordonnancement des threads par la JVM, mais ne garantit pas l'ordre d'exécution. Un thread de haute priorité est plus susceptible d'être exécuté avant un thread de basse priorité, mais cela dépend de l'implémentation de la JVM et du système d'exploitation.

Exercice 3

1. **Concept de multithreading en Java et avantages :**
   - **Multithreading** : Technique permettant l'exécution concurrente de plusieurs threads dans un programme.
   - **Avantages** :
     - Meilleure utilisation des ressources CPU.
     - Amélioration de la performance pour les tâches parallèles.
     - Réduction du temps de réponse dans les applications interactives.
2. **Programme Java simple avec deux threads, producteur et consommateur :**

```
java
Copier le code
public class ProducerConsumer {
    private static final Object LOCK = new Object();
    private static int count = 0;
    private static final int LIMIT = 10;

    public static void main(String[] args) {
        Thread producer = new Thread(() -> {
            try {
                while (true) {
                    synchronized (LOCK) {
                        while (count == LIMIT) {
                            LOCK.wait();
                        }
```

```
                    count++;
                    System.out.println("Produced: " + count);
                    LOCK.notifyAll();
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    });

    Thread consumer = new Thread(() -> {
        try {
            while (true) {
                synchronized (LOCK) {
                    while (count == 0) {
                        LOCK.wait();
                    }
                    System.out.println("Consumed: " + count);
                    count--;
                    LOCK.notifyAll();
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    });

    producer.start();
    consumer.start();
  }
}
```