Les méthodes run() et start():

- **run()** : Exécute le code dans le **thread courant**. Pas de nouveau thread créé.
- **start()** : Crée un **nouveau thread** et exécute **run()** dans ce nouveau thread.

Exemple complet avec start() et run() :

```
public class MyThread extends Thread {
  @Override
  public void run() {
    System.out.println("Thread is running...");
  }
}

public class Main {
  public static void main(String[] args) {
    MyThread thread = new MyThread();

    // Si vous utilisez run(), le code s'exécutera dans le thread principal.
    // thread.run();  // Ceci NE crée PAS de nouveau thread.

    // start() crée un nouveau thread et exécute run() dans ce nouveau thread.
    thread.start();   // Ceci crée un nouveau thread parallèle.

    System.out.println("Main thread continues...");
  }
}
```

Résumé :

- **run()** : Méthode où vous définissez la tâche du thread, mais **ne crée pas un thread séparé** si vous l'appelez directement.
- **start()** : Crée un **nouveau thread** et exécute **run()** dans ce thread, permettant ainsi un traitement parallèle.Ainsi, **on utilise start() pour réellement lancer un thread** et permettre une exécution simultanée du code, tandis que **run()** n'est qu'une méthode ordinaire contenant le code à exécuter par le thread.

TP1_2:

programme Java dans le fichier TP1_2, où un second thread est créé pour effectuer la somme des nombres entre deux entiers donnés par l'utilisateur:

```
package TPsSE;
```

- **Déclaration du package** : Le programme fait partie du package TPsSE.

```
import java.util.Scanner;
```

- **Importation de la classe Scanner** : Permet d'utiliser la classe Scanner pour lire l'entrée de l'utilisateur via la console.

```
public class TP1_2 {
```

- **Déclaration de la classe TP1_2** : Cette classe contient la logique principale du programme.

Création du thread pour la somme

- Commentaire indiquant que le thread principal va créer un second thread pour effectuer la somme.

```
  public static class thread_sum extends Thread {
```

- **Déclaration de la classe interne thread_sum** : Cette classe hérite de la classe Thread et définit un nouveau thread qui calculera la somme.

```
    String thread_name;
     int a, b, res;
```

- **Attributs de la classe thread_sum** :
  - thread_name: Le nom du thread, utile pour distinguer plusieurs threads.
  - a et b: Les bornes de la somme (les deux nombres donnés par l'utilisateur).
  - res: Le résultat de la somme calculée.

```
    public thread_sum(String name, int a, int b) {
        this.thread_name = name;    // to make difference between multiple created
threads
        this.a = a;
```

```
    this.b = b;
}
```

- **Constructeur du thread** : Ce constructeur initialise le nom du thread (thread_name), ainsi que les bornes de la somme (a et b).

```
public int sum(int a, int b) {
    int s = 0;

    for (int i = a; i <= b; i++) {
        s += i;
    }

    return s;
}
```

- **Méthode sum** : Cette méthode effectue la somme des entiers de a à b. Elle est

```
@Override
        public void run() {
            res = sum(a, b);
        }
```

- **Méthode run** : Cette méthode est exécutée lorsque le thread démarre (via l'appel à T1.start() dans le main). Ici, la somme des nombres entre a et b est calculée et stockée dans res.

```
public String getResult() {
    return "result of " + this.thread_name + " = " + res;
}
```

- **Méthode getResult** : Retourne une chaîne de caractères contenant le nom du thread et le résultat de la somme calculée.

Méthode main

```
public static void main(String[] args) throws InterruptedException {
```

- **Méthode principale main** : Point d'entrée du programme. La méthode lève une exception InterruptedException pour gérer les interruptions possibles pendant l'attente d'un thread.

```
Scanner scanner = new Scanner(System.in);
System.err.println("Give your First number a ");
```

```
        int a = scanner.nextInt();

        System.err.println("Give your Second number b ");
        int b = scanner.nextInt();
```

- **Lecture des entrées utilisateur** : Le programme demande à l'utilisateur d'entrer deux nombres a et b, qui seront utilisés pour la somme.

```
    thread_sum T1 = new thread_sum("T1", a, b);
```

- **Création du thread T1** : Un nouvel objet thread_sum est créé avec le nom "T1", et les deux nombres a et b sont passés comme paramètres.

```
    T1.start();
```

- **Démarrage du thread T1** : L'exécution du thread commence. La méthode run() de T1 est exécutée en parallèle du thread principal.

```
    Thread.sleep(2000); // wait for thread to terminate its execution
```

- **Pause du thread principal** : Le thread principal attend 2 secondes pour laisser T1 terminer son exécution. Cette attente peut être remplacée par T1.join() qui attend la fin d'exécution de T1 explicitement.

```
    System.out.println(T1.getResult());
```

- **Affichage du résultat** : Le thread principal affiche le résultat calculé par T1 en appelant la méthode getResult().

```
    }
```

- **Fin de la méthode main**.

```
}
```

- **Fin de la classe TP1_2**.