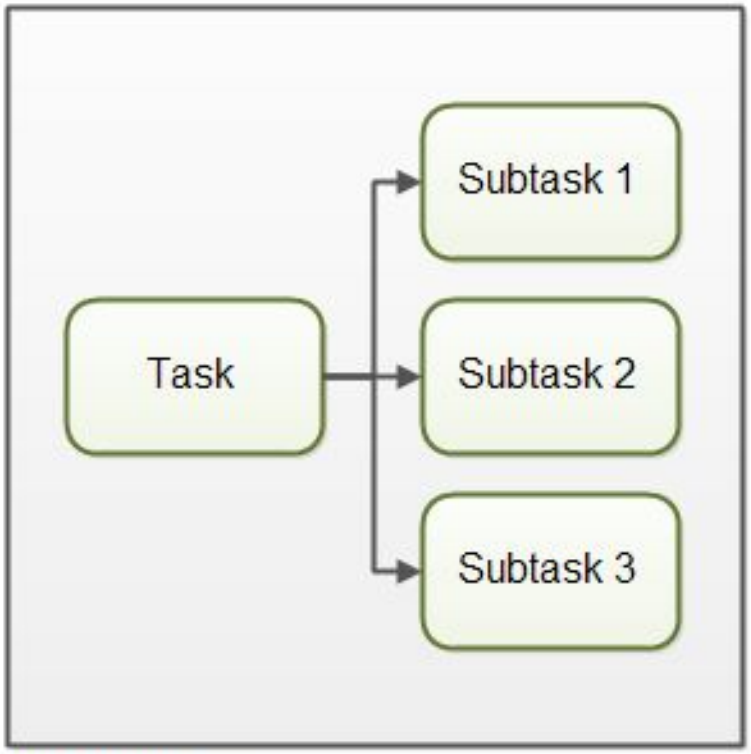
Багатопотоковість і розподілені процеси в Java

Конкурентність чи паралелізм (Concurrency vs. Parallelism)

Concurrency:
Multiple tasks makes progress at the same time.



Parallelism:
Each task is broken into subtasks which can be processed in parallel.

- Конкурентність пов'язана з тим, як додаток обробляє кілька завдань. Додаток може обробляти одну задачу в певний момент часу (послідовно) або працювати над декількома завданнями одночасно (паралельно).

- Паралелізм пов'язаний з тим, як додаток обробляє кожну окрему задачу. Додаток може обробляти завдання послідовно від початку до кінця, або розділити задачу на підзадачі.

Створення потоків в Java

- Створення потоку в Java здійснюється наступним чином:

```
Thread thread = new Thread();
```

- Для того, щоб запустити потік Java потрібно викликати його метод start():

```
thread.start();
```

Підклас Thread

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

Створення і запуск потоку:

```
MyThread myThread = new MyThread();
myTread.start();
```

Можна також створити анонімний підклас:

```
Thread thread = new Thread(){
        public void run(){
            System.out.println("Thread Running");
        }
}

thread.start();
```

Реалізація інтерфейсу Runnable

```
public class MyRunnable implements Runnable {
        public void run(){
                System.out.println("MyRunnable running");
         }
}

Thread thread = new Thread(new MyRunnable());
thread.start();
```

Анонімна реалізація Runnable:

```
Runnable myRunnable = new Runnable(){
        public void run(){
                System.out.println("Runnable running");
        }
}
Thread thread = new Thread(myRunnable);
thread.start();
```

Метод Thread.currentThread()

```
Thread thread = Thread.currentThread();

String threadName = Thread.currentThread().getName();
```

Приклад:

```
public class ThreadExample {
    public static void main(String[] args){
System.out.println(Thread.currentThread().getName());
        for(int i = 0;  i < 10;  i++){
            new Thread("" + i){
                public void run(){
                System.out.println("Thread: " + getName() + " running");
                }
                }.start();
        }
    }
}
```

Race Conditions and Critical Sections

Стан гонки є особливим станом, який може відбутися всередині критичної секції.

Критична секція являє собою фрагмент коду, який виконується декількома потоками і де послідовність виконання для потоків робить різницю в результаті одночасного виконання критичної секції.

Critical Sections

```
public class Counter {
        protected long count = 0;
        public void add(long value){
                this.count = this.count + value;
        }
}
```

this.count = 0;
A: Reads this.count into a register (0)
B: Reads this.count into a register (0)
B: Adds value 2 to register
B: Writes register value (2) back to memory. this.count now equals 2
A: Adds value 3 to register
A: Writes register value (3) back to memory. this.count now equals 3

Запобігання стану гонки

Для запобігання виникнення стану гонки необхідно переконатися в тому, що критична секція виконується атомарно. Це означає, що як тільки один потік виконує її, ніякі інші потоки не можуть виконати її, поки перший потік не залишить критичну секцію.

```
public class TwoSums {
    private int sum1 = 0;
    private int sum2 = 0;
    public void add(int val1, int val2){
        synchronized(this){
            this.sum1 += val1;
                this.sum2 += val2;
        }
    }
}
```

```
public class TwoSums {
    private int sum1 = 0;
    private int sum2 = 0;
    public void add(int val1, int val2){
synchronized(this){ this.sum1 += val1; }
synchronized(this){ this.sum2 += val2; }
    }
}
```

Thread Safety

- Local Variables:

```
public void someMethod(){
        long threadSafeInt = 0;
        threadSafeInt++;
}
```

- Local Object References:

```
public void someMethod(){
        LocalObject localObject = new LocalObject();
localObject.callMethod();
        method2(localObject);
}
public void method2(LocalObject localObject){
localObject.setValue("value");
}
```

- Object Member Variables:

```
public class NotThreadSafe{
    StringBuilder builder = new StringBuilder();
    public add(String text){
        this.builder.append(text);
    }
}
```

```
NotThreadSafe sharedInstance = new NotThreadSafe();
new Thread(new MyRunnable(sharedInstance)).start();
new Thread(new MyRunnable(sharedInstance)).start();
public class MyRunnable implements Runnable{
NotThreadSafe instance = null;
    public MyRunnable(NotThreadSafe instance){
        this.instance = instance;
    }
    public void run(){
        this.instance.add("some text");
    }
}
```

```
new Thread(new MyRunnable(new NotThreadSafe())).start();
new Thread(new MyRunnable(new NotThreadSafe())).start();
```