# Programação Concorrente







# DMPUTAÇÃO (GUFCG

# java.util.concurrent Package

Disponibiliza classes utilitárias úteis para desenvolvimento em programação concorrente. Este pacote inclui algumas pequenas estruturas extensíveis padronizadas, bem como algumas classes que fornecem funcionalidade útil e são, de outra forma, tediosas ou difíceis de implementar. Aqui estão breves descrições dos principais componentes.





# Simplificando o Gerenciamento de Threads

## ExecutorService

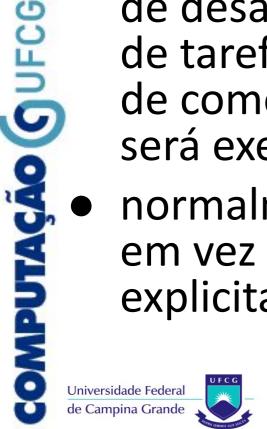
- Um objeto que executa tarefas enviadas
- Fornece uma maneira de desacoplar o envio de tarefas da mecânica de como cada tarefa será executada
- normalmente usado em vez de criar threads explicitamente

```
RunnableTask implements Runnable {
    public void run() {
        System.out.println("Asynchronous task");
    }
}

new Thread(new RunnableTask()).start()
new Thread(new RunnableTask()).start()
new Thread(new RunnableTask()).start()
```

```
ExecutorService executorService =
Executors.newFixedThreadPool(10);
```

```
executorService.execute(new RunnableTask());
executorService.execute(new RunnableTask());
executorService.execute(new RunnableTask());
```



## Criando um ExecutorService

ExecutorService executorService = Executors.newSingleThreadExecutor();

Cria um Executor que usa uma única thread operando em uma fila ilimitada. Se esse thread única terminar devido a uma falha durante a execução antes do desligamento, uma nova tomará seu lugar, se necessário, para executar tarefas subsequentes.



## Criando um ExecutorService

ExecutorService executorService = Executors.newCachedThreadPool();

Cria um Executor que cria novas threads conforme necessário, mas reutilizará threads construídas anteriormente quando estiverem disponíveis.



## Criando um ExecutorService

ExecutorService executorService = Executors.newFixedThreadPool(10);

Cria um pool de threads que utiliza um número fixo de threads operando em uma fila ilimitada compartilhada. A qualquer momento, no máximo n threads estarão ativas processando tarefas. Se tarefas adicionais forem enviadas quando todas as threads estiverem ativas, elas esperam na fila até que uma thread esteja disponível.



# OMPUTAÇÃO (C) UFCG

# Delegando uma tarefa para um ExecutorService

- void execute(Runnable)
- Future<?> submit(Runnable)
- Future<T> submit(Callable<T>)

Qual a diferença entre Runnable e Callable?



# DMPUTAÇÃO (CIUFCG

# Delegando uma tarefa para um ExecutorService

- void execute(Runnable)
- Future<?> submit(Runnable)
- Future<T> submit(Callable<T>)

## Qual a diferença entre Runnable e Callable?

```
Runnable runnableTask = () -> {
    try {
        TimeUnit.MILLISECONDS.sleep(300)
    ;
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
};
Universidade Federal de Campina Grande
```

```
Callable<String> tarefa = () -> {
    Thread.sleep(300);
    return "Resultado da tarefa";
};
```

## Future

Um Future representa o resultado de uma computação assíncrona. O resultado só pode ser recuperado usando o método get quando a computação for concluída, bloqueando se necessário até que esteja pronto



# OMPUTAÇÃO (GUFCG

### **Future**

Um Future representa o resultado de uma computação assíncrona. O resultado só pode ser recuperado usando o método get quando a computação for concluída, bloqueando se necessário até que esteja pronto

```
Future<String> resultado = executor.submit(tarefa);
System.out.println("Aguardando resultado...");
System.out.println("Resultado: " + resultado.get());
```



# OMPUTAÇÃO (GUFCG

## Finalizando um ExecutorService

## shutdown()

 Ele espera até que a execução de todas as tarefas enviadas seja concluída.

## shutdownNow()

• Ele imediatamente encerra todas as tarefas em execução/pendentes.



# COMPUTAÇÃO (GUFCG

