02

A leitura e saída e como se testar variações inusitadas

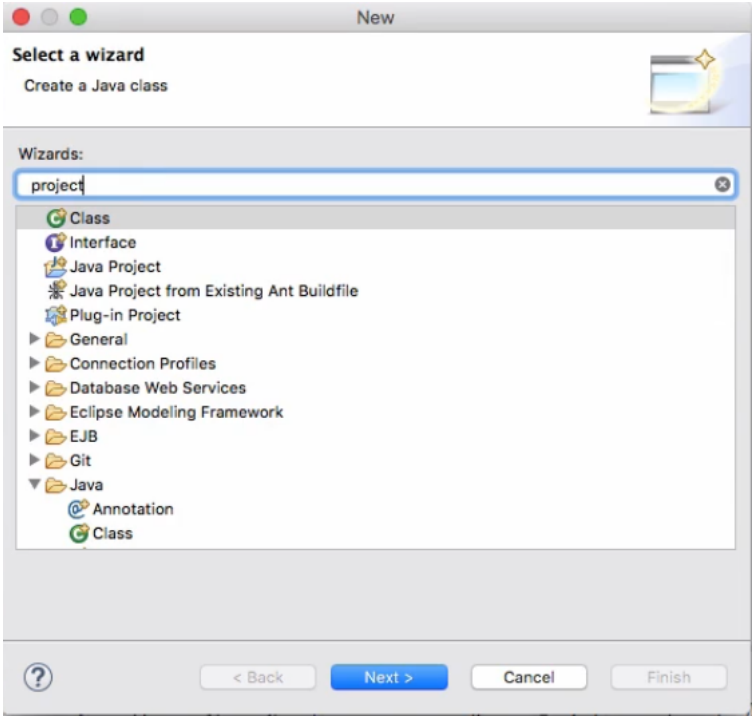
Transcrição

Já sabe por onde podemos começar a atacar o nosso problema? Já falamos que quando temos um problema, ele nos dá três coisas para fazer:

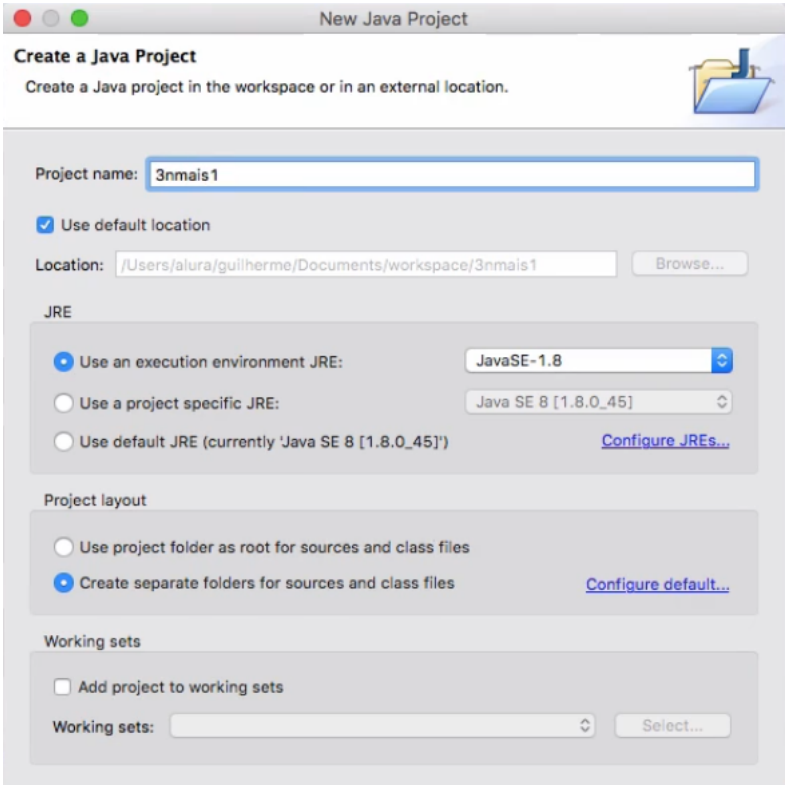
1. Ler as informações que o cliente nos deu;
2. Rodar o algoritmo, seja ele simples ou complexo;
3. Devolver a saída.

Podemos cuidar simultaneamente de um pouquinho de cada uma, ou de uma de cada vez. Mas, como já conversamos, é preciso fazer um trabalho baseado nos nossos exemplos que o cliente deu.

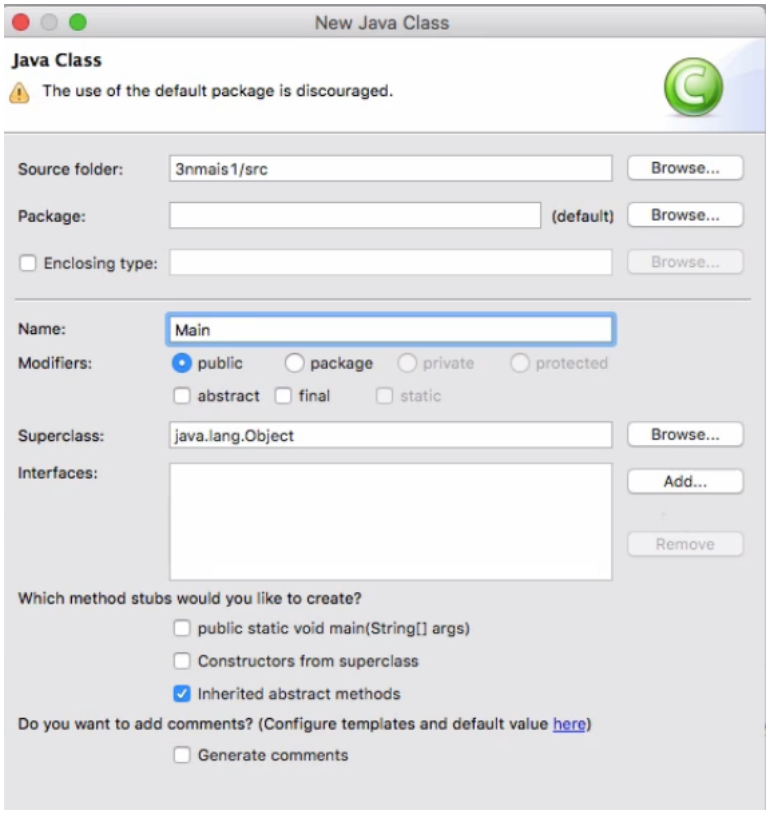
Assim, vamos criar um projeto no Eclipse para começarmos a trabalhar. Fecharemos tudo o que estiver aberto e usaremos `Ctrl + N` para criar o novo projeto Java.



O projeto se chamará `3nmnais1` .



A seguir, criaremos a classe `Main`, em um pacote sem nome.



Agora já podemos criar o método `main`.

```
public class Main {

  public static void main(String[] args){

  }
}
```

Já podemos criar a `entrada1.txt`, clicando com o botão direito no pacote e em `New > File`, baseada no exemplo dado pelo cliente.

```
1 10
100 200
201 210
900 1000
```

Se esse exemplo não funcionar, nada irá. Sabemos que a saída deve repetir os números da entrada, e depois adicionar o número que corresponde ao maior comprimento do ciclo. Vamos primeiro ter certeza de que o programa lerá a entrada corretamente e imprimirá os mesmos números na saída. Primeiro, criaremos um `Scanner`, que lerá da entrada padrão `System.in`, e deverá ler até o final do arquivo. Por isso, usaremos o `hasNextLine`.

```
public class Main {

  public static void main(String[] args){
      Scanner scanner = new Scanner(System.in);
      while(scanner.hasNextLine()){

      }

  }
}
```

Temos que pegar os dois números (`int`). O enunciado chama de `i` o menor número e de `j` o maior? Temos que tomar cuidado com as nossas palavras. Ele chamou o primeiro de `i`, e o segundo de `j`.

```
public class Main {

  public static void main(String[] args){
      Scanner scanner = new Scanner(System.in);
      while(scanner.hasNextLine()){
        int i = scanner.nextInt();
        int j = scanner.nextInt();
      }

  }
}
```

Agora, acrescentaremos um `sysout` com os espaços entre os números. Como ainda não rodamos o algoritmo, colocaremos o terceiro como `" xxxxxxx"`.

```
public class Main {

  public static void main(String[] args){
      Scanner scanner = new Scanner(System.in);
      while(scanner.hasNextLine()){
        int i = scanner.nextInt();
        int j = scanner.nextInt();
        System.out.println(i + " " + j + " xxxxxxx");
      }
```

```
    }
  }
```

Vamos testar para ver se está funcionando? Testaremos no terminal, lembrando que é assim que o cliente fará. Começaremos entrando no diretório correto.

```
Alura-Azul:bin alura$ cd ..
Alura-Azul:palincod alura$ cd..
Alura-Azul:workspace alura$ cd
.metadata/              3nmais1/              palincod/
.recommenders/          RemoteSystemsTempFile/ test/
Alura-Azul:workspace alura$ cd 3nmais1/
```

Então, criaremos o diretório `bin` e testaremos nossa `entrada1.txt`, no `Main`.

```
Alura-Azul:bin alura$ cd ..
Alura-Azul:palincod alura$ cd..
Alura-Azul:workspace alura$ cd
.metadata/              3nmais1/              palincod/
.recommenders/          RemoteSystemsTempFile/ test/
Alura-Azul:workspace alura$ cd 3nmais1/
Alura-Azul:3nmais1 alura$ cd bin
Alura-Azul:bin alura$ ls
Main.class        entrada1.txt
Alura-Azul:bin alura$java Main < entrada1.txt
```

Limparemos a tela e vamos rodar.

```
Alura-Azul:bin alura$java Main < entrada1.txt
1 10 xxxxxxx
100 20 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
```

No lugar do `xxxxxxx` precisamos colocar o maior comprimento de ciclo entre os números anteriores. Assim, precisaremos calcular o comprimento de ciclo do `1`, do `2`, do `3` ... Calma aí, vamos calcular o comprimento de ciclo de um número primeiro?

Mas já sabemos que a leitura veio certa, e a saída também saiu corretamente. O próximo passo seria começar a implementar o algoritmo, mas... A entrada está ok mesmo? São só essas as entradas válidas? Que entradas o cliente não mencionou, mas seria interessante testar?

Vá pensando nessas entradas à medida que vamos conversando. Quando estivermos implementando, podemos considerar entradas em que ainda não tínhamos pensado. Eu costumo fazer as duas coisas: releio o enunciado para ver o que é uma entrada válida para esse problema, e ter certeza de que estou cobrindo todos os casos, e depois, enquanto trabalho no algoritmo, tento encontrar as entradas capciosas.

Revendo o enunciado:

> The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 10,000 and greater than 0.

Teremos um par de inteiros por linha, e eles podem ser menores que `1000000` e maiores que `0`. Ou seja, podemos ter o `0`? Não, porque ele falou que os números serão **maiores** que `0`. Da mesma maneira, o maior número que teremos será `999999`.

Então vamos testar. Criaremos a `entrada2.txt`, para separar dos exemplos. Eu gosto de deixar separados, mas podemos começar o arquivo repetindo os exemplos que já testamos, para garantir que continuarão funcionando.

```
1 10
100 200
201 210
900 1000
```

Sabemos que `999999` é um número válido. Vamos fazer do `999000` até o `999999`?

```
1 10
100 200
201 210
900 1000
999000 999999
```

> You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j. You can assume that no operation overflows a 32-bit integer.

Temos que processar todos os comprimentos entre `i e j`, e não podemos dar *overflow* na operação, que não pode exceder `32 bits`. À medida que escrevermos o algoritmo, tentaremos pensar em algum outro caso, inclusive com algumas palavras que já mencionamos.

O próximo passo é ter certeza de que estamos lendo essa entrada direito.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

Parece tudo certo. Note que os pares dessa entrada estão apresentados em ordem crescente. Vamos colocar os mesmos números duas vezes, para garantir que nosso algoritmo funcione sem uma ordem determinada. Afinal, provavelmente o cliente jogará dados de maneira aleatória. Assim, a `entrada2.txt` fica:

```
1 10
100 200
201 210
900 1000
999000 999999
1 10
```

```
100 200
201 210
900 1000
999000 999999
```

Ainda estamos deixando um caso para trás. Mas será que a saída desse fica certa?

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
1 10 xxxxxxx
100 200 xxxxxxx
201 210 xxxxxxx
900 1000 xxxxxxx
999000 999999 xxxxxxx
```

Por enquanto está tudo certo. A entrada é lida corretamente e a saída também está certa. A questão agora é o algoritmo. Vamos trabalhar nele daqui a pouco. Até lá!