

Programação Orientada Pelos Objectos

Asserções



Programa

2



O que é uma asserção

3

- Mecanismo que permite ao programador verificar se determinado pressuposto é respeitado
- Ao ler as instruções de asserção no código, sabemos que determinada condição no ponto da asserção deve ser sempre satisfeita
- Durante a execução do programa
 - O programa permite inferir se as asserções estabelecidas são verdadeiras ou não
 - Se um asserção não for verdadeira, é lançado uma exceção da classe `AssertionError`, que é derivada de `Error`

Activação de asserções

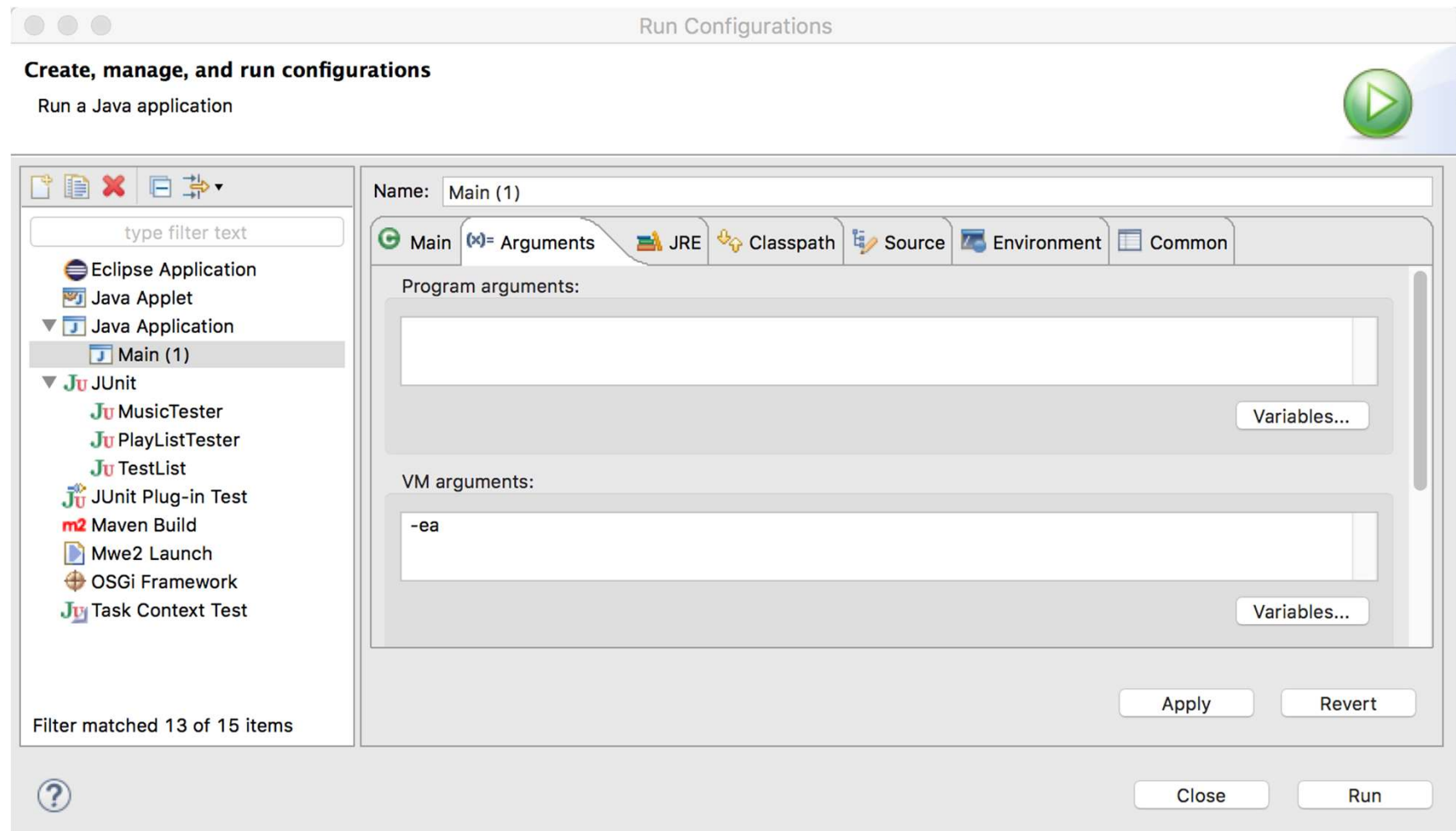
4

- O utilizador do programa tem sempre a possibilidade de desactivar o mecanismo de verificação de asserções
- As asserções devem ser usadas durante o desenvolvimento para detectar erros mas em geral devem ser “desligadas” quando o programa é disponibilizado aos utilizadores
- Compilação
 - Com mecanismo de asserção
javac -source MyProgram.java
 - Sem mecanismo de asserção
javac MyProgram.java
- Activação de asserções

java -enableassertions MyProgram.java (ou -ea)

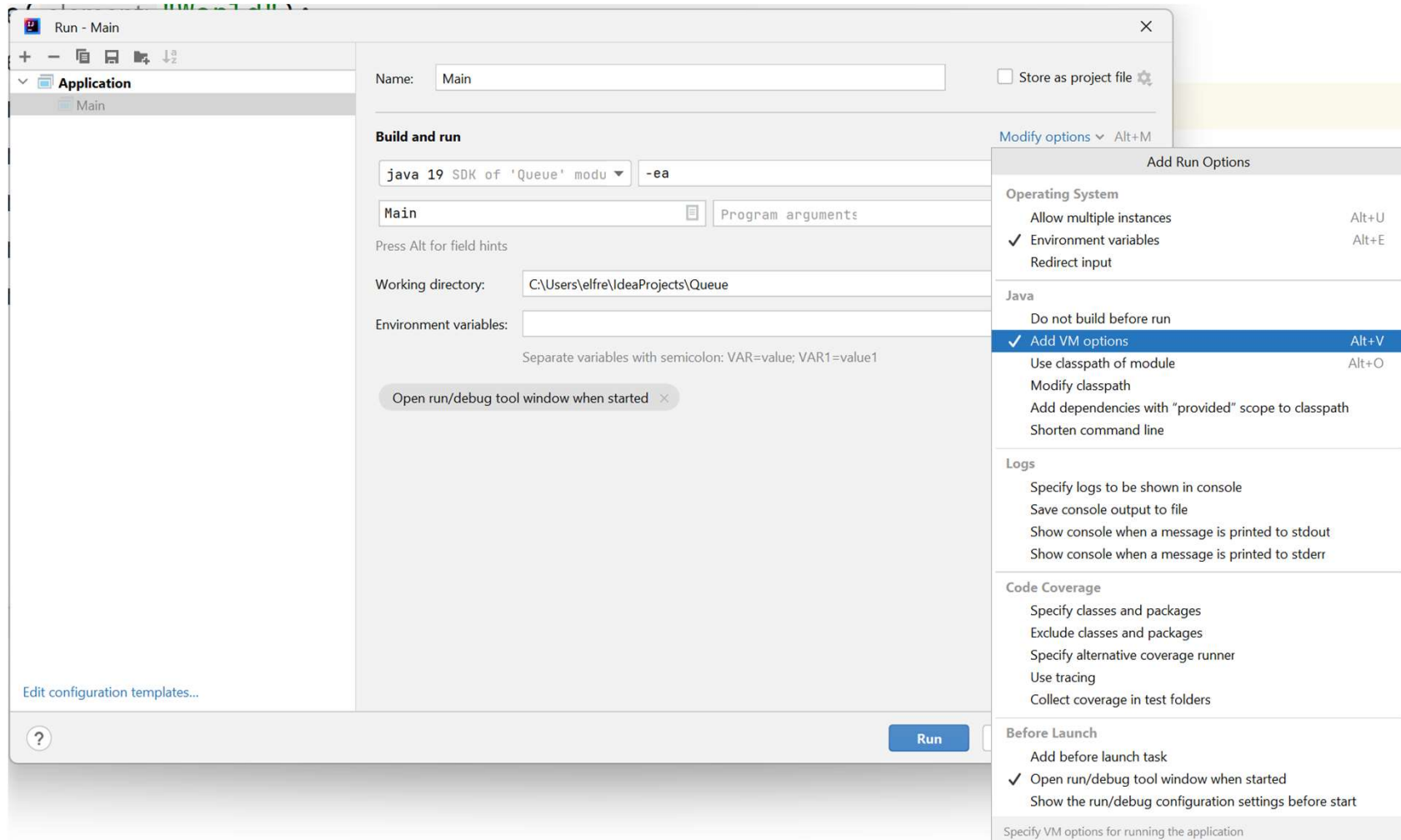
Activação de asserções no Eclipse

5



Activação de asserções no IntelliJ

6



Formatos de definição de asserções

7

○ Primeiro formato

○ **assert** *Expressão* ;

- em que *Expressão* é uma expressão booleana
- Quando o sistema corre a instrução, avalia a *Expressão*
 - Se for falsa, o sistema lança a exceção `AssertionError`, sem qualquer informação adicional

○ Segundo formato

○ **assert** *Expressão1*: *Expressão2* ;

- em que *Expressão1* é uma expressão booleana e *Expressão2* uma expressão que tem obrigatoriamente um valor (que não pode ser **void**)
- Usa-se este segundo formato quando se pretende dar informação adicional ao lançar a exceção `AssertionError`
 - A representação como `String` do valor calculado por *Expressão2* é usada como mensagem de erro detalhada da violação da asserção

Como escolher o formato da asserção?

8

- A mensagem de erro é interna, e não para o utilizador
- O objectivo da mensagem de erro é ajudar a perceber exactamente o que provocou a violação da asserção, para que o problema possa ser resolvido
 - Normalmente, a mensagem é lida juntamente com o Stack Trace da excepção levantada
- Apenas se deve usar o segundo formato quando temos informação adicional realmente útil que complementa o stack trace
 - Por exemplo, se a asserção envolve a relação entre dois valores x e y , uma expressão representando a mensagem de erro potencialmente interessante poderia ser:
 - `"x: " + x + ", y: " + y`

Quando acrescentar asserções:

9

Sem asserções

```
if (i % 3 == 0) {  
    ...  
} else if (i % 3 == 1) {  
    ...  
} else if (i % 3 == 2) {  
    ...  
} else { //Sabemos que i é negativo  
}
```

Com asserções

```
if (i % 3 == 0) {  
    ...  
} else if (i % 3 == 1) {  
    ...  
} else if (i % 3 == 2) {  
    ...  
} else {  
    assert i < 0: i;  
    ...  
}
```

Quando acrescentar asserções:

10

Sem asserções

```
if (i % 3 == 0) {  
    ...  
} else if (i % 3 == 1) {  
    ...  
} else { //Sabemos que(i % 3 == 2)  
    ...  
}
```

Com asserções

```
if (i % 3 == 0) {  
    ...  
} else if (i % 3 == 1) {  
    ...  
} else {  
    assert i % 3 == 2 : i;  
    ...  
}
```

Repare que o código sem asserções pode falhar. Por exemplo, se *i* for negativo, o resto da divisão também seria negativo, nesse caso. Sem a asserção, não detectaríamos o problema!

Quando acrescentar asserções:

11

Sem asserções

```
switch(suit) {  
  case Suit.CLUBS: ...  
    break;  
  case Suit.DIAMONDS: ...  
    break;  
  case Suit.HEARTS: ...  
    break;  
  case Suit.SPADES: ...  
}
```

A alternativa do lançamento de excepções oferece protecção mesmo quando as asserções estiverem desligadas.

Com asserções

```
switch(suit) {  
  case Suit.CLUBS: ...  
    break;  
  case Suit.DIAMONDS: ...  
    break;  
  case Suit.HEARTS: ...  
    break;  
  case Suit.SPADES: ...  
    break;  
  default:  
    assert false: suit;  
    // ou, em vez do assert  
    // throw new AssertionError(suit);  
}
```

Quando acrescentar asserções:

12

Sem asserções

```
void foo() {  
    for (...) {  
        if (...)  
            return;  
    }  
    // Nunca chega aqui!  
}
```

Com asserções

```
void foo() {  
    for (...) {  
        if (...)  
            return;  
    }  
    // Nunca chega aqui!  
    assert false;  
}
```

Cuidado especial a ter: esta técnica **não deve ser usada** se o próprio compilador do Java for capaz de detectar automaticamente que o código não é atingível. E o compilador consegue detectar esta situação, em muitos casos (mas não em todos).

Quando acrescentar asserções:

Pré-Condições, pós-condições e invariantes de classe

13

- Uma pré-condição é uma condição que se tem de verificar **antes** da execução de uma operação
 - Por convenção, violações a pré-condições de métodos públicos devem dar origem ao lançamento de exceções apropriadas
 - Apenas devemos usar asserções para testar pré-condições de métodos **privados**

```
/**
 * Ajusta o intervalo de refrescamento, que tem de ser legal.
 * @param interval intervalo de refrescamento, em milisegundos.
 */
private void setRefreshInterval(int interval) {
    assert interval > 0 && interval <= 1000/MAX_REFRESH_RATE : interval;
    ... //Ajusta o intervalo de refrescamento
}
```

Quando acrescentar asserções:

Pré-Condições, **pós-condições** e invariantes de classe

14

- Uma pós-condição é uma condição que se tem de verificar **depois** da execução de uma operação
 - Podemos testar pós-condições relativas a métodos com qualquer visibilidade

```
/**
 * Retorna o perímetro de um quadrado.
 * @param l é a largura
 * @param c é o comprimento
 * @return l * c.
 * @throws ArithmeticException, se o resultado não for o esperado.
 */
public double perimeter(int l, int c) {
    double result = 2 * l + 2 * c;
    assert result == 2 * (l + c); // Exemplo exageradamente simples.
    return result;
}
```

Quando acrescentar asserções:

Pré-Condições, pós-condições e **invariantes de classe**

15

- Um invariante de classe é uma condição que se tem de verificar **sempre** em qualquer objecto dessa classe, em qualquer estado que se encontre
 - Excepto em estados transitórios entre dois estados consistentes, tipicamente durante a execução de um método
 - Mas tem de se verificar quer **antes**, quer **depois** da execução do método
 - Exemplo: o saldo de uma conta tem de ser sempre positivo ou zero

```
// Returns true if the balance greater than amount
public boolean canWithdraw(double amount) {
    return balance > amount;
}

// Antes do retorno de qualquer método público, incluindo o construtor
// poderíamos incluir a seguinte asserção. Mas, em geral, não é
// necessário testar também no início do método.
assert canWithdraw(0);
```


Quando não acrescentar asserções

16

- Verificação de argumentos em métodos públicos
 - Recorde que as asserções podem ser ligadas ou desligadas e que o programa deve funcionar bem sempre, **mesmo que elas estejam desligadas**
 - Os métodos públicos fazem parte do “contrato” de uma classe, que tem de ser obedecido, com ou sem excepções
 - Argumentos errados devem dar origem a uma excepção apropriada, por exemplo:
 - `IllegalArgumentException`
 - `IndexOutOfBoundsException`
 - `NullPointerException`
 - O insucesso da asserção “esconde” a excepção, o que dificulta a correcta detecção e eliminação do problema

Quando não acrescentar asserções

17

- Na realização de tarefas do programa obrigatórias para a sua correcta execução
 - Como as asserções podem estar inibidas, isso resultaria na não execução desse código fundamental!

○ Exemplo de **como não fazer**:

```
//Errado! Acção do programa definida como parte da asserção  
assert names.remove(null);
```

○ Como resolver o problema:

```
//Correcto! A acção precede a asserção  
boolean nullsRemoved = names.remove(null);  
assert nullsRemoved; // Este código funciona mesmo que  
// as asserções estejam inibidas
```

Exercício

18

- Implemente a interface `Queue<E>`, que representa uma fila genérica, com as operações
 - `void enqueue(E elem)`
 - `E dequeue()`
 - `boolean isEmpty()`
 - `int size()`
- Use asserções para garantir que o seu código está correcto