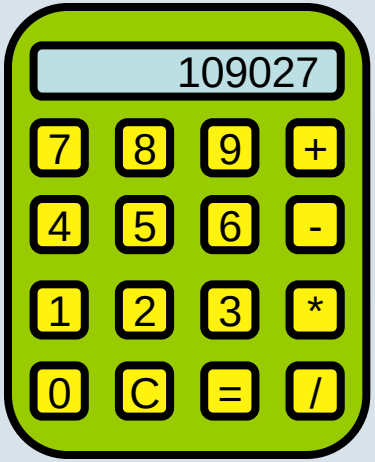
# Calculadora

# Calculadora

- Defina em Java uma classe `Calculator` cujos objectos representam uma máquina de calcular muito simples (apenas com as quatro operações básicas).
- Programe a sua classe no BlueJ.
- Teste um (ou vários) objectos `Calculator`, e verifique se se comportam da forma esperada.


# Calculadora

- Os métodos de cada objecto calculadora executam a acção associada à pressão de cada tecla na máquina
- O valor inteiro devolvido por cada método representa o número (inteiro) visualizado no visor da máquina
- A máquina tem um visor e as teclas apresentadas



# Calculadora

- Operações reconhecidas

```
int numberkey(int digit)
```
Pressão de uma tecla numérica ( `digit` = 1,2,3,4,5,6,7,8,9 ou 0)

```
int plus()
```
Pressão da tecla +

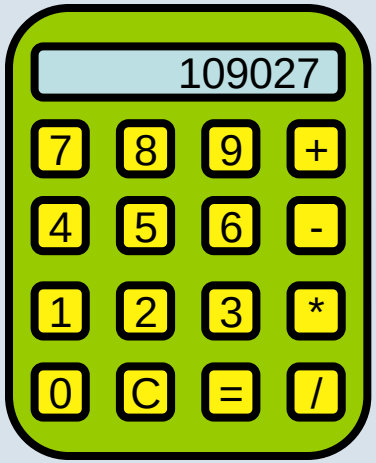
```
int minus()
```
Pressão da tecla -

```
int times()
```
Pressão da tecla x

```
int divide()
```
Pressão da tecla /

```
int equals()
```
Pressão de tecla =

```
int clear()
```
Pressão da tecla C



# Calculadora

```
Calculator texas = new Calculator();
texas.numberkey(2)
2    (int)
texas.numberkey(0)
20   (int)
texas.plus()
20   (int)
texas.numberkey(3)
3   (int)
texas.numberkey(0)
30 (int)
texas.equals()
50   (int)
texas.clear()
0   (int)
```

# Calculator: execução da operação

```
...
if (operator == PLUS)
    firstOperand = firstOperand + secondOperand;
else if (operator == MINUS)
        firstOperand = firstOperand - secondOperand;
     else if (operator == TIMES)
             firstOperand = firstOperand * secondOperand;
          else firstOperand =
                     firstOperand / secondOperand;
...
```

# Instrução switch

**A instrução de selecção ou condicional**

```
switch (expression) {
    case literal-1: statements-1 break;
    case literal-2: statements-2 break;
    . . // (more cases) .
    case literal-N: statements-N break;
    default: // optional default case
                    statements-(N+1)
} // end of switch statement
```

- **expression** tem de ser do **tipo int ou char**.
- O código executado é o que segue o literal que corresponde ao valor guardado em **expression**.
- A utilização de **break** é opcional e põe termo à execução da instrução **switch**. Se for omitido, todos os casos a partir do escolhido são executados (se nenhum contiver o **break**).
- A utilização do **default** também é opcional. Este ramo é executado se nenhum dos casos explícitos corresponder ao valor de **expression**.
- O bloco de instruções de um caso pode ser vazio. Isto implica a execução do mesmo código para mais do que um literal.

# Execução da operação com switch

```
...
switch (operator) {
    case PLUS :
          firstOperand = firstOperand + secondOperand;
          break;
    case MINUS:
          firstOperand = firstOperand - secondOperand;
          break;
    case TIMES:
          firstOperand = firstOperand * secondOperand;
          break;
    default:
          firstOperand = firstOperand / secondOperand;
          break;
}
...
```

# Execução da operação com switch

```
...
switch (operator) {
    case PLUS :
          firstOperand = firstOperand + secondOperand;
          break;
    case MINUS:
          firstOperand = firstOperand - secondOperand;
          break;
    case TIMES:
          firstOperand = firstOperand * secondOperand;
          break;
    default:
          firstOperand = firstOperand / secondOperand;
          break;
}
...
```

Isto funciona, mas faz algum sentido considerar o operador / como o operador por omissão? Ou outro qualquer, neste caso?

# Execução da operação com switch

```
...
switch (operator) {
    case PLUS :
          firstOperand = firstOperand + secondOperand;
          break;
    case MINUS:
          firstOperand = firstOperand - secondOperand;
          break;
    case TIMES:
          firstOperand = firstOperand * secondOperand;
          break;
    case DIV:
          firstOperand = firstOperand / secondOperand;
          break;
    default:
          break;
}
...
```

Esta alternativa é mais elegante! Neste caso, não queremos nenhum comportamento por omissão, mas veremos outros exemplos em que faz sentido definir esse comportamento.

# Execução da operação com switch

```
...
switch (operator) {
    case PLUS :
          firstOperand = firstOperand + secondOperand;
          break;
    case MINUS:
          firstOperand = firstOperand - secondOperand;
          break;
    case TIMES:
          firstOperand = firstOperand * secondOperand;
          break;
    case DIV:
          firstOperand = firstOperand / secondOperand;
          break;
    default:
          break;
}
...
```

Se não queremos nenhum comportamento por omissão, não vale a pena deixar o tratamento para o caso default. Lembre-se, o default é opcional!

# Execução da operação com switch

```
...
switch (operator) {
    case PLUS :
          firstOperand = firstOperand + secondOperand;
          break;
    case MINUS:
          firstOperand = firstOperand - secondOperand;
          break;
    case TIMES:
          firstOperand = firstOperand * secondOperand;
          break;
    case DIV:
          firstOperand = firstOperand / secondOperand;
          break;
}
...
```

Esta alternativa é a mais elegante!

# O que acontece se removermos um break?

```
...
switch (operator) {
    case PLUS :
          firstOperand = firstOperand + secondOperand;
    case MINUS:
          firstOperand = firstOperand - secondOperand;
          break;
    case TIMES:
          firstOperand = firstOperand * secondOperand;
          break;
    case DIV:
          firstOperand = firstOperand / secondOperand;
          break;
}
...
```

Se neste exemplo removermos o break do caso PLUS, quando o operador seleccionado for o PLUS, as operações executadas são todas até ao break seguinte:

```
firstOperand = firstOperand + secondOperand;
firstOperand = firstOperand - secondOperand;
```

# Expressão tem de ser int ou char

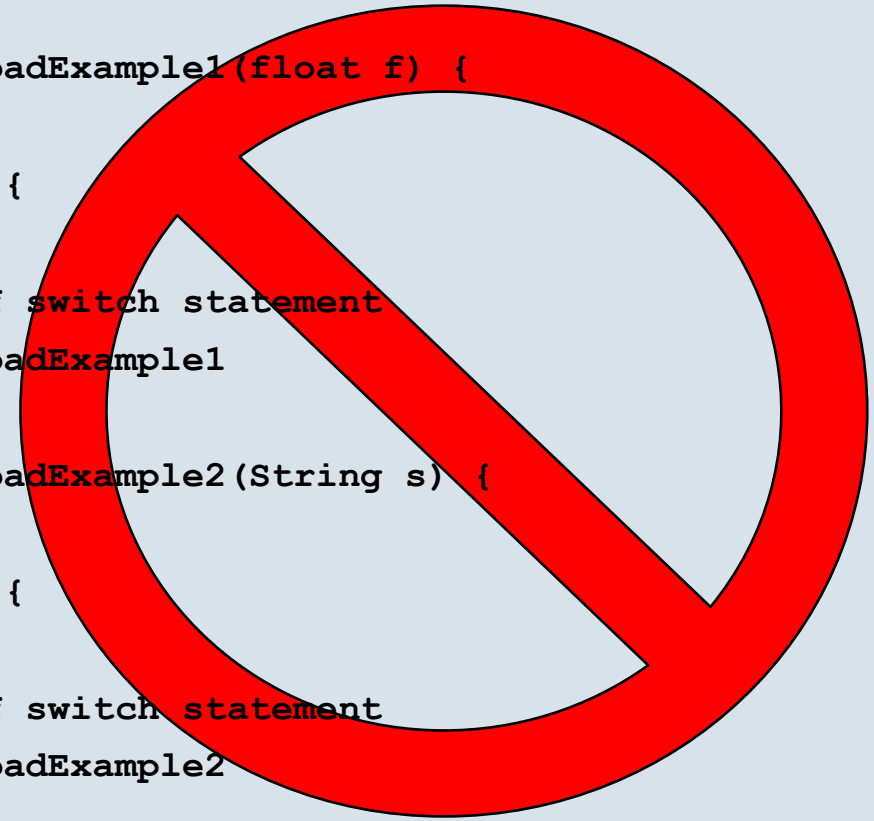
- A expressão expression tem de ser dos tipos int, ou char, e constante
  - Exemplos

```
public void goodExample1(int i) {
  ...
  switch (i) {
    ...
  } // end of switch statement
} // end of goodExample1

public void goodExample2(char c) {
  ...
  switch (c) {
    ...
  } // end of switch statement
} // end of goodExample2
```

# Expressão tem de ser int ou char

- A expressão expression tem de ser dos tipos int, ou char, e constante
  - Exemplos de erros típicos

```
public void badExample1(float f) {
  ...
  switch (f) {
    ...
  } // end of switch statement
} // end of badExample1

public void badExample2(String s) {
  ...
  switch (s) {
    ...
  } // end of switch statement
} // end of badExample2
```



Em ambos os casos, o compilador dá erro, porque o tipo da expressão não é nem **int** nem **char!**

# Os literais de cada case são constantes

- Exemplo correcto (o literal 3 é constante):

```
public void goodExample(int i) {
  ...
  switch (i) {
    case 3:...
    ...
  } // end of switch statement
} // end of goodExample
```

- Exemplo errado (x não é constante!):

```
public void badExample(int i) {
  int x = /* uma expressão variável */;
  switch (i) {
     case x: ...
     ...
  } // end of switch statement
} // end of badExample
```

# O corpo de um case pode ser vazio

- Isto permite juntar mais que um caso

```
public void emptyCases(int i) {
  ...
  switch (i) {
    case 1:
    case 3:
    case 5:
       /* do something for cases 1, 3 and 5 */
       break;
    case 2:
       /* do something else for case 2 */
       break;
  } // end of switch statement
  ...
} // end of emptyCases
```

# Programas com Decisões

- Neste capítulo, vimos como definir em Java programas que tomam decisões e executam acções em alternativa como resultado de verificar condições.




- Foram introduzidas as seguintes noções importantes:
  - A instrução condicional: **`if-then-else`**
  - O bloco de instruções (ou instrução composta)
  - A instrução de selecção: **`switch`**