Controlling Program Flow

Hendrik Speleers

Controlling Program Flow

Università di Roma Tor Vergata

- Overview
  - Mathematical expressions
    - Basic arithmetic operations
    - Type casting
  - Boolean expressions
    - Logical and relational operations
  - Selection statements
    - If-then, If-then-else, switch
  - Repetition statements
    - While, do-while, for

Controlling Program Flow

Università di Roma Tor Vergata

- Variables: primitive types
  - Same syntax and operations as in C++

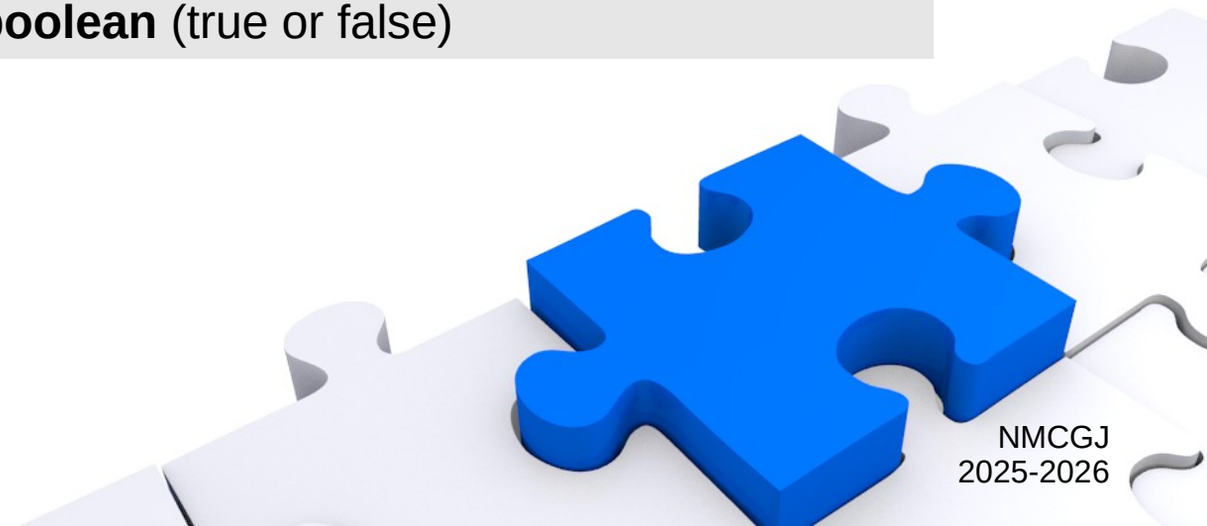
integers

Type	Min value	Max value
byte	$-2^7 = -128$	$2^7-1 = 127$
short	$-2^{15} = -32768$	$2^{15}-1 = 32767$
int	$-2^{31} \approx -2 \cdot 10^9$	$2^{31}-1 \approx 2 \cdot 10^9$
long	$-2^{63} \approx -9 \cdot 10^{18}$	$2^{63}-1 \approx 9 \cdot 10^{18}$

reals

Type	Min value	Max value
float	$\approx -3{,}4 \cdot 10^{38}$	$\approx 3{,}4 \cdot 10^{38}$
double	$\approx -1{,}7 \cdot 10^{308}$	$\approx 1{,}7 \cdot 10^{308}$
char (Unicode)		
boolean (true or false)		

  - Special memory treatment
    - In the stack
    - Size is machine independent (not in C++)



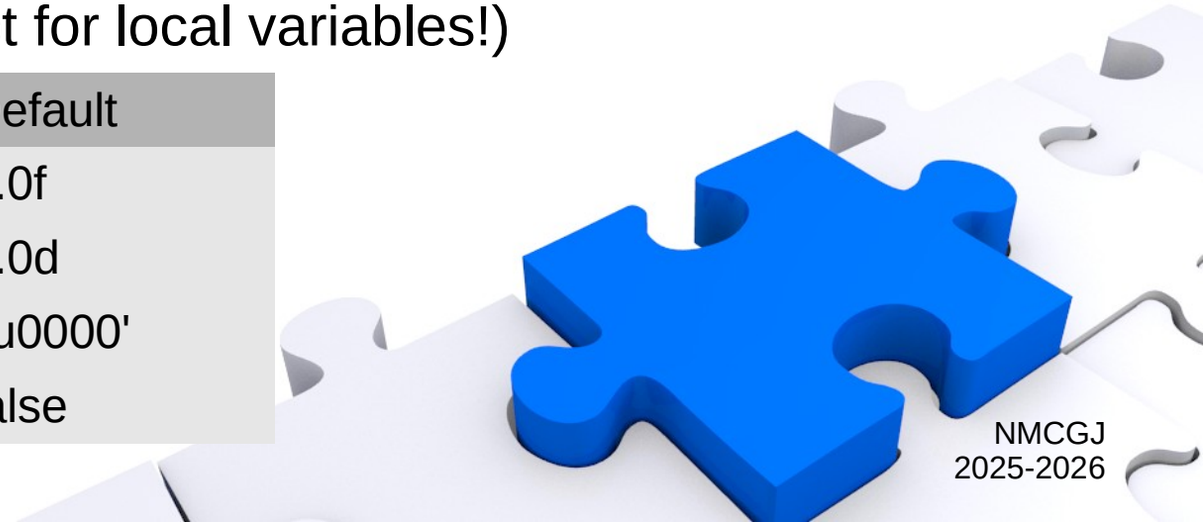
Controlling Program Flow

Università di Roma Tor Vergata

- Variables: primitive types
  - Related classes
    - Wrapper classes to create objects (read only!)
      - Byte, Short, Integer, Long, Float, Double, Character, Boolean
    - Classes for performing high-precision arithmetic
      - BigInteger, BigDecimal
  - Default values (only for class variables, not for local variables!)

Type	Default
byte	(byte)0
short	(short)0
int	0
long	0L

Type	Default
float	0.0f
double	0.0d
char	'\u0000'
boolean	false

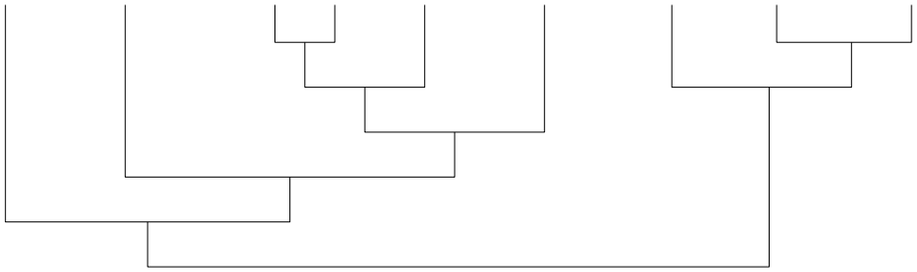


Controlling Program Flow

Università di Roma Tor Vergata

- Mathematical expressions
  - Basic arithmetic operations
    - Addition (+), subtraction (−), multiplication (*), division (/), modulus (%)
  - Priority rules: high >> low
    - (x) >> −x >> x * y, x / y, x % y >> x + y, x − y

a * (b + (− c / d) / e) * (f − g % h)



Controlling Program Flow

Università di Roma Tor Vergata

- Mathematical expressions
  - Basic arithmetic operations
    - Addition (+), subtraction (−), multiplication (*), division (/), modulus (%)
  - Priority rules: high >> low
    - (x) >> −x >> x * y, x / y, x % y >> x + y, x − y
  - Auto increment and decrement

```
int x = 10;
x = x - 1;
x -= 1;
x--; --x;
```

```
int x = 10;
x = x + 1;
x += 1;
x++; ++x;
```

Controlling Program Flow

- Mathematical expressions
  - Basic arithmetic operations
    - Addition (+), subtraction (−), multiplicatio (%)
  - Priority rules: high >> low
    - (x) >> −x >> x * y, x / y, x % y >>
  - Auto increment and decrement

```
int x = 10;
x = x - 1;
x -= 1;
x--; --x;
```

```
int x = 10;
x = x + 1;
x += 1;
x++; ++x;
```

post-increment

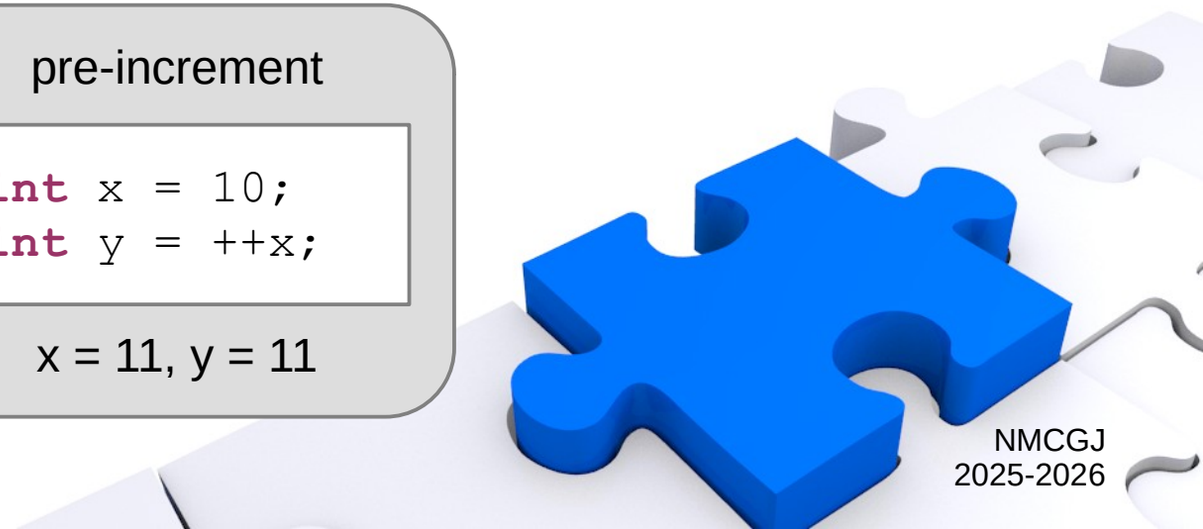
```
int x = 10;
int y = x++;
```

x = 11, y = 10

pre-increment

```
int x = 10;
int y = ++x;
```

x = 11, y = 11



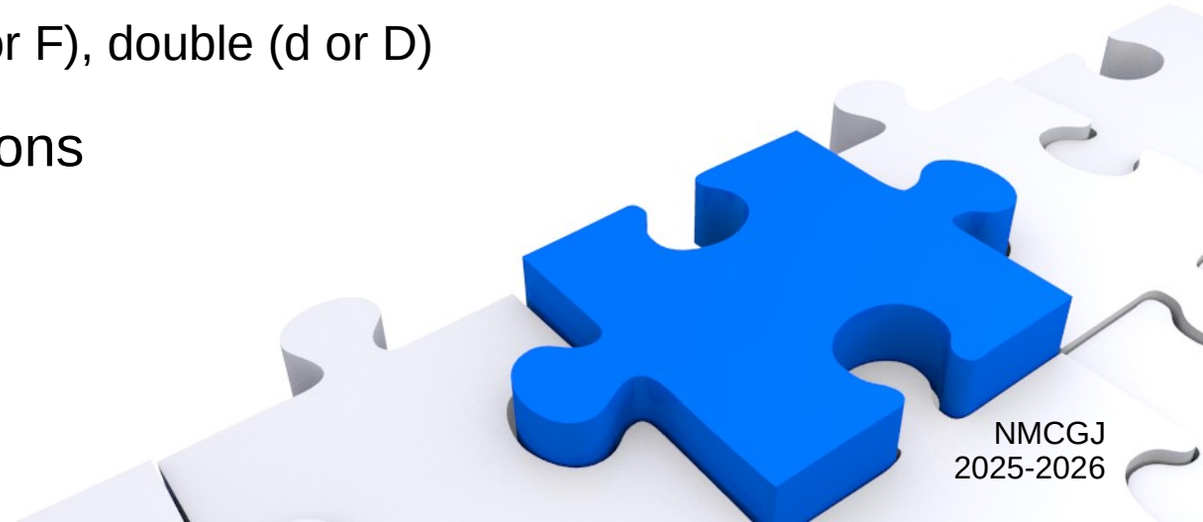
Controlling Program Flow

- Type casting: conversion of one type into another
  - Type cast operations

```
(<data type>) <expression> ;
```

    - Required for narrowing conversion
    - Implicit for widening conversion

      byte → short → int → long → float → double
  - Literals:
    - hexadecimal (0x or 0X), long (l or L), float (f or F), double (d or D)
  - Implicit: promotion to larger type in operations
    - int + int → int
    - long + int → long, int + double → double
    - byte + char + int → int



Controlling Program Flow

Università di Roma Tor Vergata

- Type casting: conversion of one type into another
  - Examples

```
int i = 0x2f;
short s = (short) i;
float f = 1e-27f;
```

```
int x = 10;
x / 3;
(double) x / 3;
(double) (x / 3);
```

Controlling Program Flow

Università di Roma Tor Vergata

- Type casting: conversion of one type into another
  - Examples

```
int i = 0x2f;
short s = (short) i;
float f = 1e-27f;
```

```
int x = 10;
x / 3;                  = 3
(double) x / 3;         = 3.333333
(double) (x / 3);       = 3.000000
```

Controlling Program Flow

- Type casting: conversion of one type into another
  - Casting does not necessarily mean that the content is preserved!
  - Watch out for overflow → cycling through type range
  - Example

```
byte a = 120;
a = (byte) (a + 10);
```

Controlling Program Flow

- Type casting: conversion of one type into another
  - Casting does not necessarily mean that the content is preserved!
  - Watch out for overflow → cycling through type range
  - Example

```
byte a = 120;
a = (byte) (a + 10);
```

= 120
= −126

byte range is [−128, 127]

Controlling Program Flow

- The class Math
  - Contains
    - Static methods for common math functions (sin, cos, exp, log, pow, sqrt, abs, ...)
    - Static variables for common math constants (PI, E)
  - Example
    - $z = \frac{1}{2}\sin\left(x - \frac{\pi}{\sqrt{y}}\right)$ is written as

```
double x, y, z;
z = 0.5 * Math.sin(x - Math.PI / Math.sqrt(y));
```

Controlling Program Flow

Università di Roma Tor Vergata

- Boolean expressions
  - Logical values
    - true, false (type boolean)
  - Logical operations
    - and (&&), or (||), not (!)
  - Relational operations
    - equal (==), not equal (!=), less than (<), greater than (>)
    - less than or equal (<=), greater than or equal (>=)

a	b	a && b	a || b	!a
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

```
int x, y; boolean z;
z = (x > 0 && x <= 100 || y > 0 && y <= 100);
```

Controlling Program Flow

Università di Roma Tor Vergata

- Boolean expressions
  - Short-circuiting
    - Exiting boolean expression when unambiguously determined
    - x && y : y is evaluated only if x is true
    - x || y : y is evaluated only if x is false
  - Examples

```
x != 0 && (y / x) < 20
```

no problem

```
(y / x) < 20 && x != 0
```

problem if x is zero (arithmetic exception)

Controlling Program Flow

Università di Roma Tor Vergata

- Operator precedence rules

Category	Operator
subexpression	( )
unary operators	– ++ –– !
multiplicative op.	* / %
additive operators	+ –
comparison op.	< <= > >=
equality operators	== !=
boolean and	&&
boolean or	||
assignment	=

```
x > 0 && x < 10 || x + y > 0
```

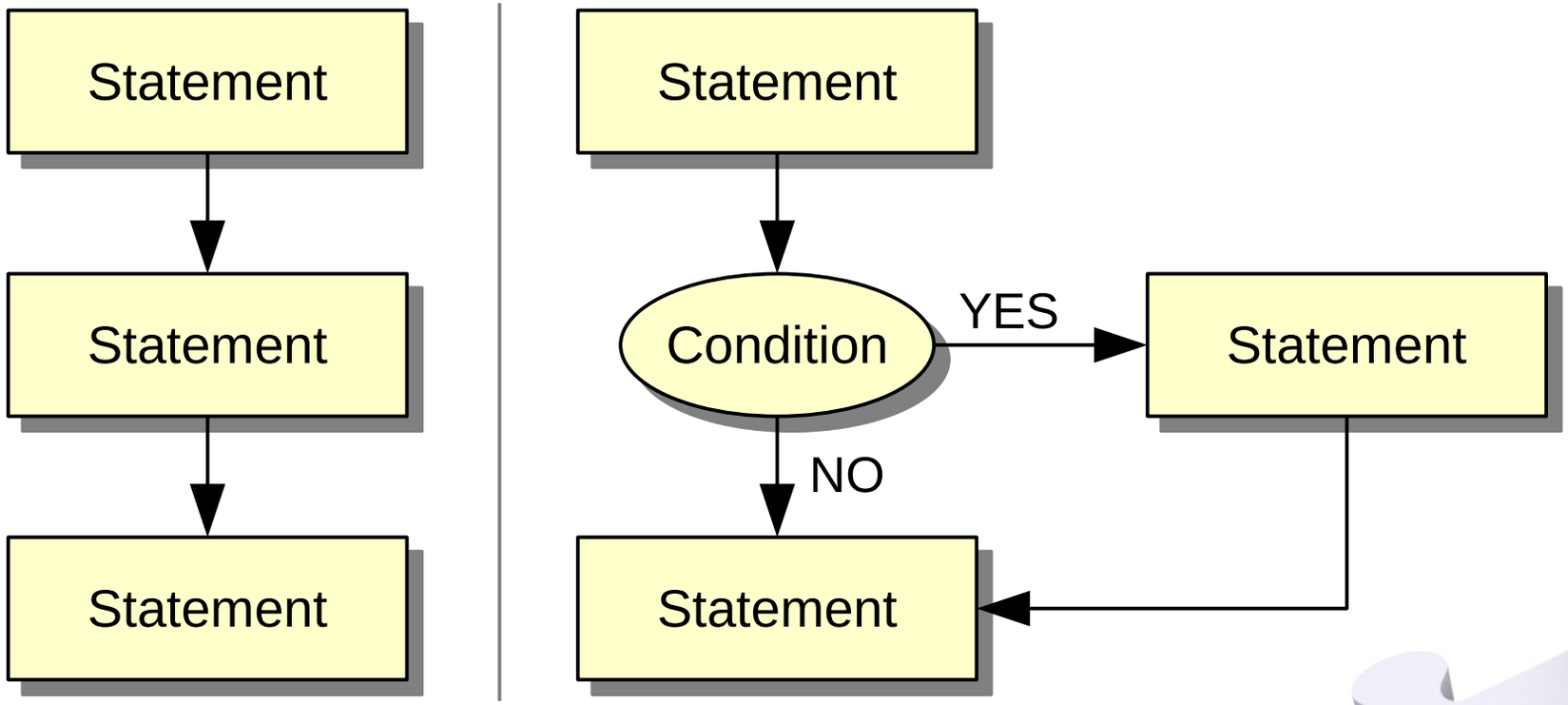
equivalent

```
((x > 0) && (x < 10)) || ((x + y) > 0)
```

Controlling Program Flow

Università di Roma Tor Vergata

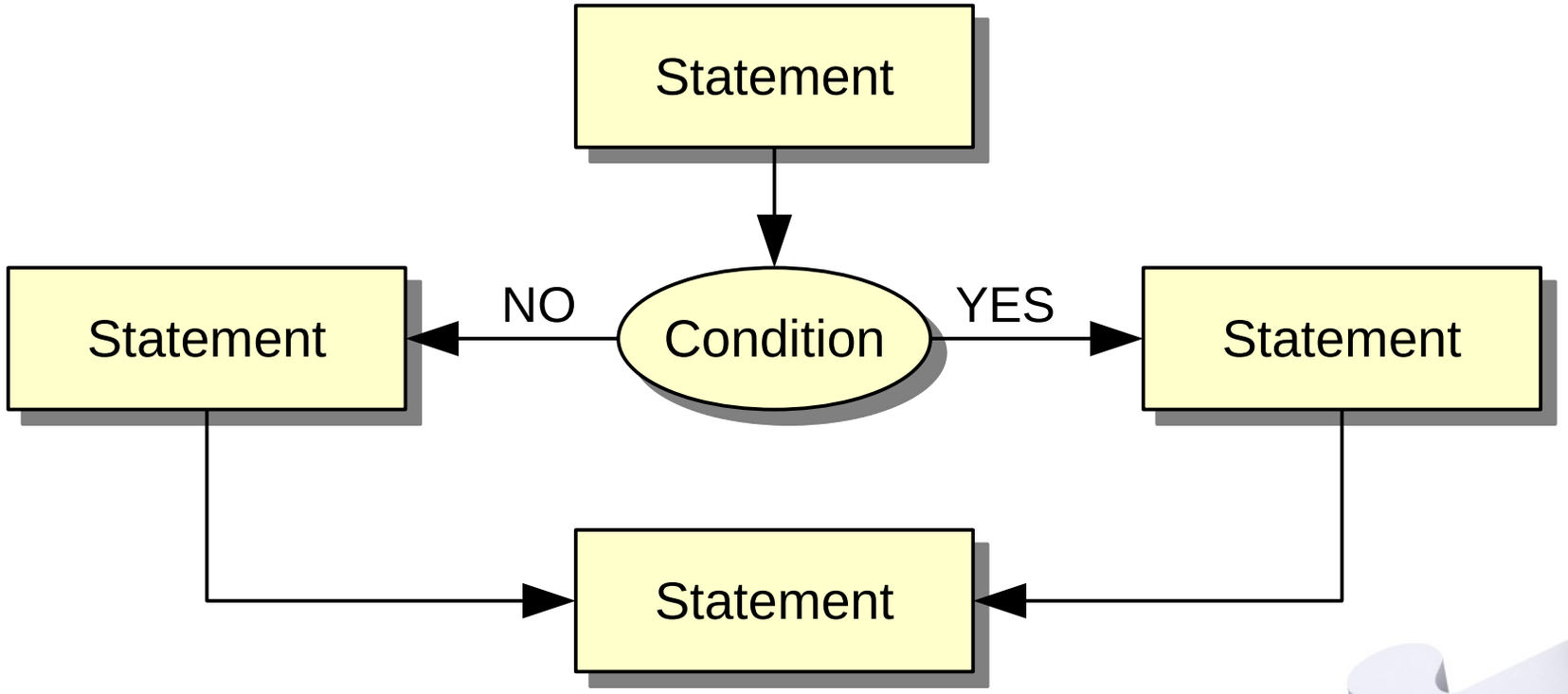
- Selection statements



Controlling Program Flow

Università di Roma Tor Vergata

- Selection statements



Controlling Program Flow

Università di Roma Tor Vergata

- Selection statements
  - If-then statement

```
if (<boolean expression>)
   <then block>
```

  - If-then-else statement

```
if (<boolean expression>)
   <then block>
else
   <else block>
```

```
public int sign(int value) {
   if (value > 0)
      return +1;
   else if (value < 0)
      return -1;
   else
      return 0;
}
```

if <block> consists of multiple statements, then use { }

Controlling Program Flow

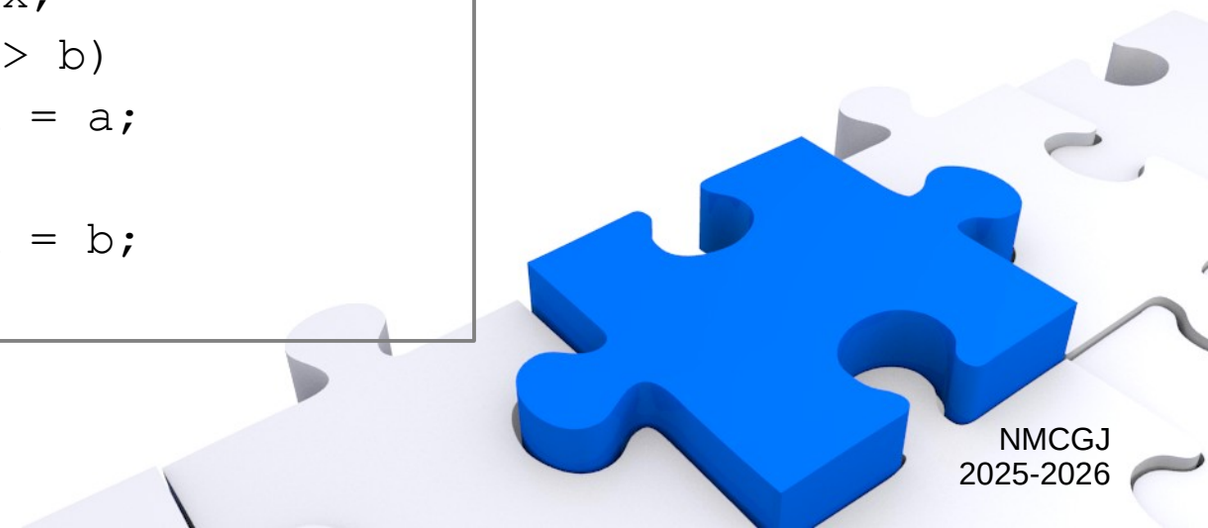
Università di Roma Tor Vergata

- Selection statements
  - Conditional operator ?

```
<boolean expression> ? <value true> : <value false>
```

```
int a = 10, b = 20;
int max = (a > b) ? a : b;
```

```
int a = 10, b = 20;
int max;
if (a > b)
   max = a;
else
   max = b;
```



Controlling Program Flow

- Selection statements
  - Switch statement

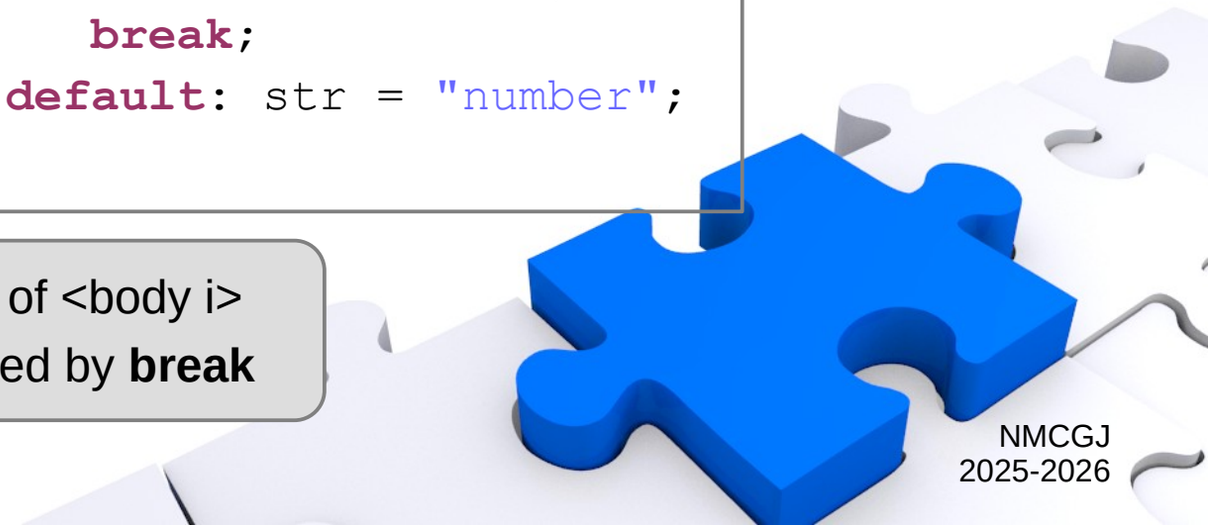
only byte, short, char, or int primitive

```
switch (<integer expr.>) {
   <case label 1> : <body 1>
   ...
   <case label n> : <body n>
}
```

```
String str;
switch (value) {
   case 0: str = "zero";
      break;
   case 1: str = "one";
      break;
   default: str = "number";
}
```

replace <case label i> by
- **case** <constant>
- **default**

execution of <body i>
is terminated by **break**



Controlling Program Flow

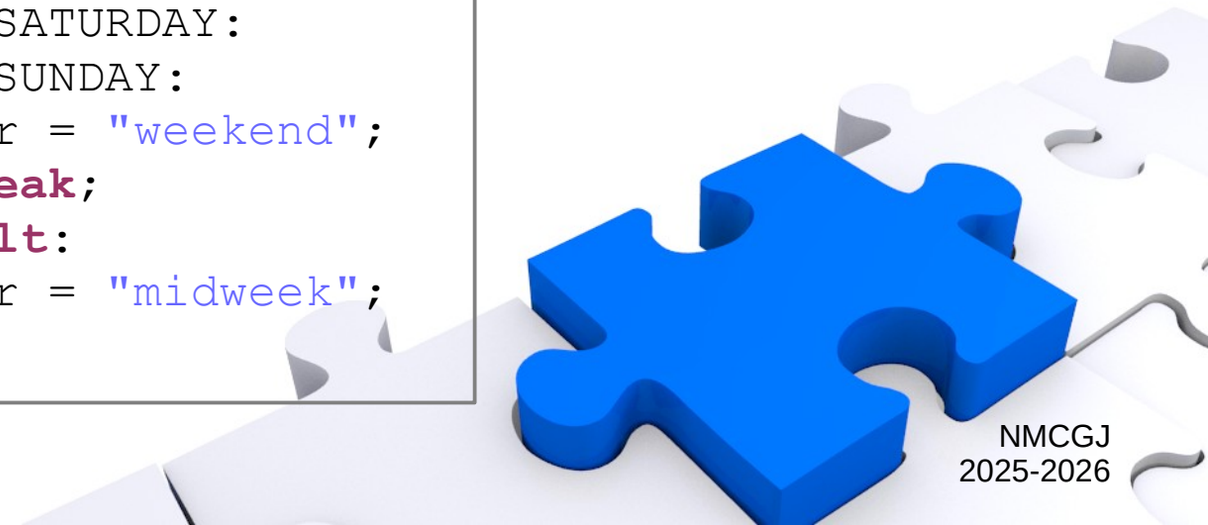
Università di Roma Tor Vergata

- Selection statements
  - Enumerated types: the **enum** keyword (since Java SE5)
    - Representation of a group of constants
    - In combination with a switch statement

```
public enum Day {
    SUNDAY, MONDAY,
    TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY,
    SATURDAY
}

Day day = Day.MONDAY;
```

```
String str;
switch (day) {
    case SATURDAY:
    case SUNDAY:
        str = "weekend";
        break;
    default:
        str = "midweek";
}
```



Controlling Program Flow

Università di Roma Tor Vergata

- Selection statements
  - Switch expression (since Java SE14)

```
switch (<integer expr.>) {
    <case lbl 1> -> <value 1>
    ...
    <case lbl n> -> <value n>
}
```

```
String str =
    switch (value) {
        case 0 -> "zero";
        case 1 -> "one";
        default -> "number";
    };
```

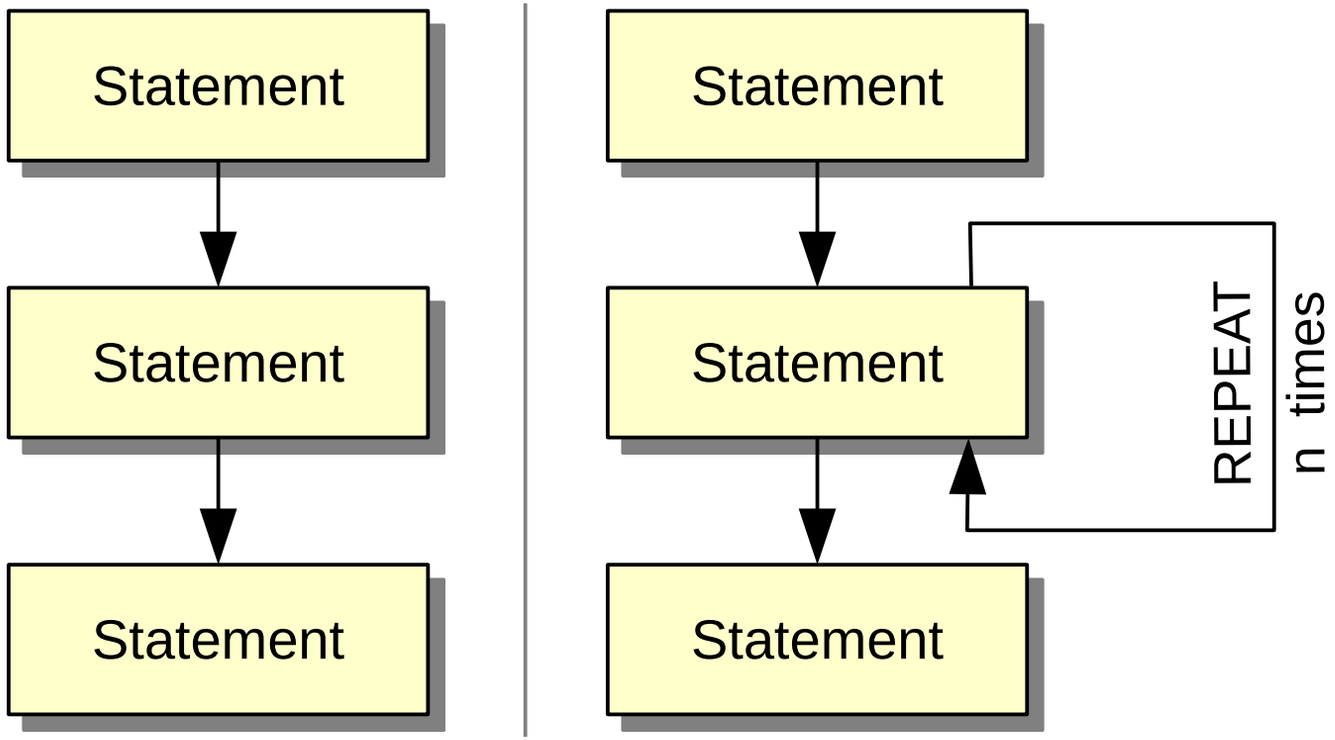
replace <case lbl i> by
- **case** <constant>
- **default**

No fall through,
no need to use **break**

Controlling Program Flow

Università di Roma Tor Vergata

- Repetition statements



Controlling Program Flow

Università di Roma Tor Vergata

- Repetition statements
  - While loop

```
while (<boolean expression>)
    <while block>
```

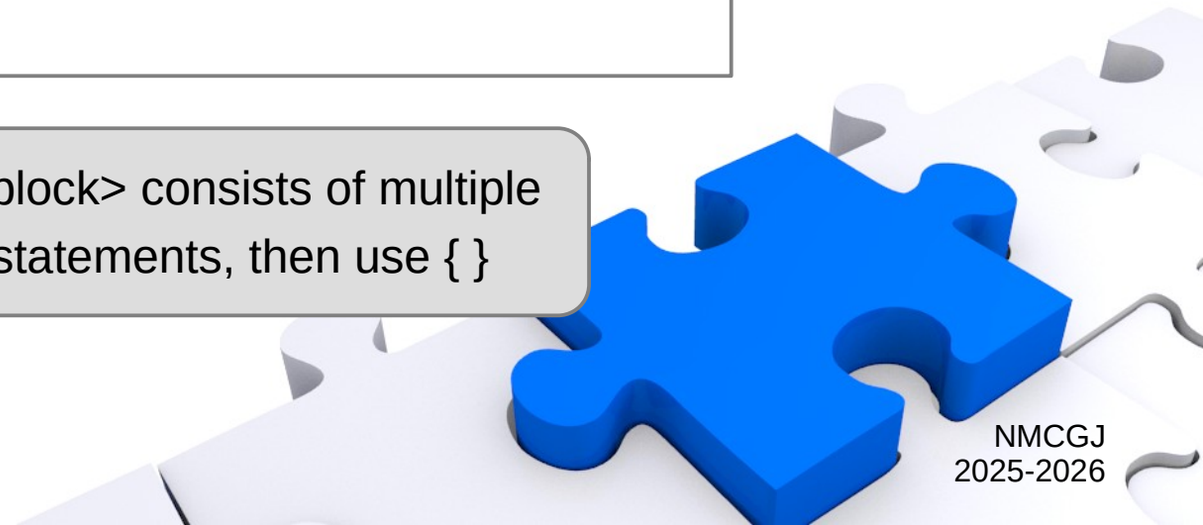
  - Do-while loop

```
do
    <while block>
while (<boolean expression>);
```

```
int s = 0, n = 0;

while (n < 20) {
    n += 1;
    s += n;
}
```

if <block> consists of multiple statements, then use { }



Controlling Program Flow

Università di Roma Tor Vergata

- Repetition statements
  - Reasoning about while loop

```
<initialization>
// invariant
while (condition) {
    // invariant && condition
    <statements>
    // invariant
}
// invariant && !condition
// => result
```

invariant: a property that is true before and after each iteration

result: a consequence of invariant and negation of condition

weak condition gives robust result

Controlling Program Flow

- Repetition statements
  - Reasoning about while loop

```
int s = 0, n = 0;

// inv: ??
while (n < 20) {

    n += 1;

    s += n;

}

// res: s == sum(1..20)
```

condition

invariant && !condition

Controlling Program Flow

Università di Roma Tor Vergata

- Repetition statements
  - Reasoning about while loop

```
int s = 0, n = 0;

// inv: s == sum(1..n) && n <= 20
while (n < 20) {
    // s == sum(1..n)    && n < 20
    n += 1;
    // s == sum(1..n-1) && n <= 20
    s += n;
    // s == sum(1..n)    && n <= 20
}
// s == sum(1..n) && n <= 20 && n >= 20
// res: s == sum(1..20)
```

Controlling Program Flow

Università di Roma Tor Vergata

- Repetition statements
  - For loop

```
for (<initialization>; <boolean expr.>; <increment>)
   <for block>
```

if <block> consists of multiple statements, then use { }

```
<initialization>;
while (<boolean expr.>) {
   <for block>
   <increment>;
}
```

additional flow control by
- **break**: stops loop
- **continue**: stops iteration

Controlling Program Flow

- Repetition statements
  - For loop

```
for (<initialization>; <boolean expr.>; <increment>)
   <for block>
```

if <block> consists of multiple statements, then use { }

```
int s = 0;

for (int n=0; n<20; n++) {
   s += (n + 1);
}
```

additional flow control by

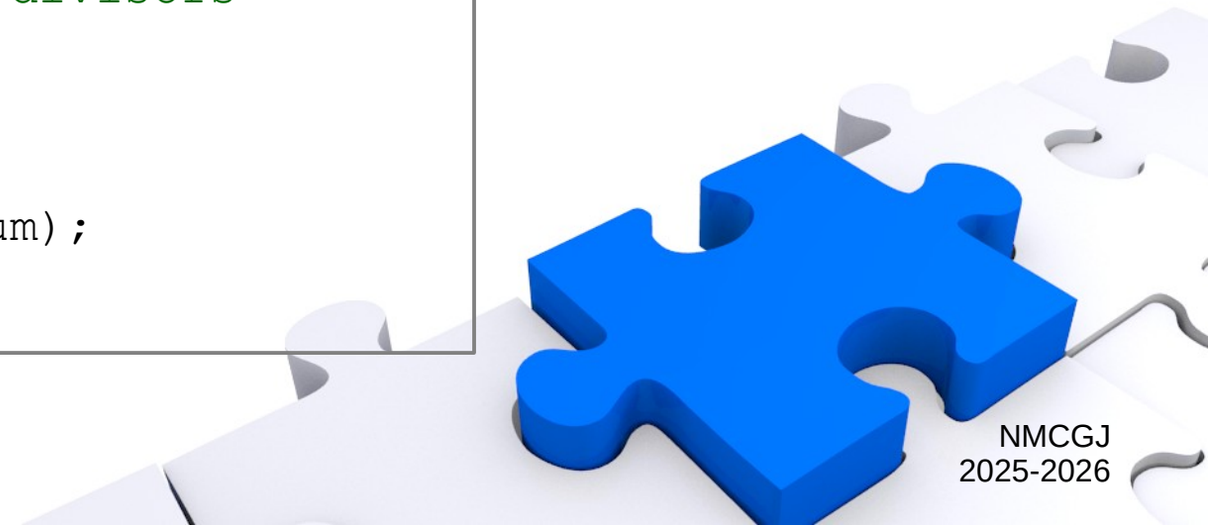
- **break**: stops loop
- **continue**: stops iteration

Controlling Program Flow

Università di Roma Tor Vergata

- Java program: flow control
  - Generating the prime numbers up to 100 (naive way)

```
int limit = 100; // define limit
for (int num = 2; num < limit; num++) {
   boolean isPrime = true;
   for (int div = 2; div < num; div++)
      if (num % div == 0) { // check divisors
         isPrime = false;
         break;
      }
   if (isPrime) System.out.println(num);
}
```

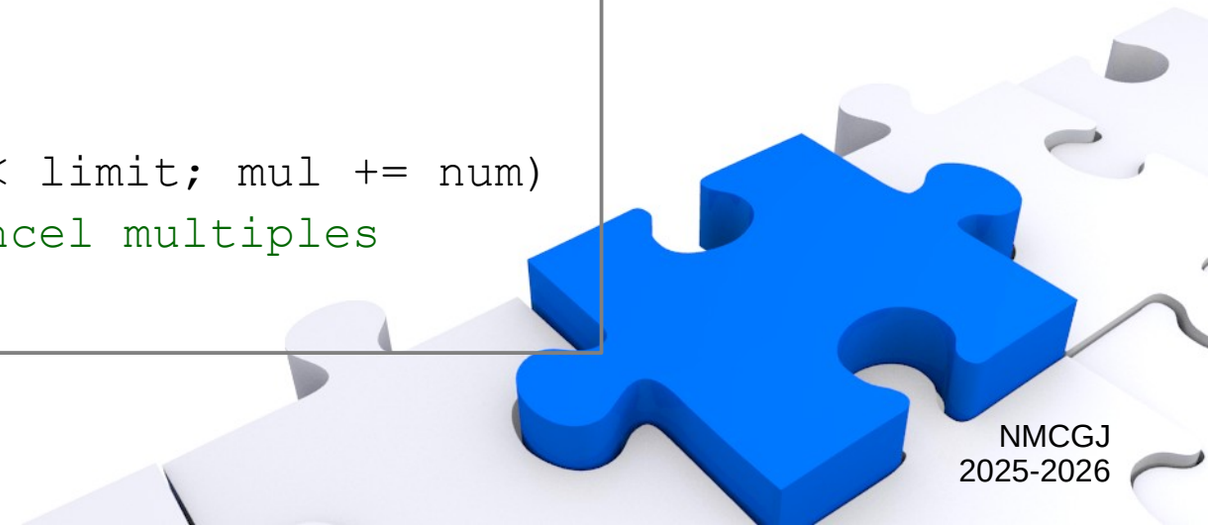


Controlling Program Flow

Università di Roma Tor Vergata

- Java program: flow control
  - Generating the prime numbers up to 100 (sieve of Eratosthenes)

```
int limit = 100; // define limit
boolean[] isPrime = new boolean[limit];
for (int num = 2; num < limit; num++)
   isPrime[num] = true;
for (int num = 2; num < limit; num++)
   if (isPrime[num]) {
      System.out.println(num);
      for (int mul = num * num; mul < limit; mul += num)
         isPrime[mul] = false; // cancel multiples
   }
```



Controlling Program Flow

Università di Roma Tor Vergata

- Good programming style

> The goal of a good style is to make your code more readable for you and for others!

  - Some rules:
    - Commenting & Documentation
    - Consistent and meaningful names
    - Consistent indentation / whitespaces
    - Avoiding of duplication / redundancy
    - ...

Controlling Program Flow

Università di Roma Tor Vergata

- Good programming style
  - Consistent and meaningful names

```
String s1, s2;
int t;
```

versus

```
String firstName, lastName;
int temperature;
```

Follow standard name conventions

Controlling Program Flow

Università di Roma Tor Vergata

- Good programming style
  - Consistent indentation / whitespaces

```
public                                  class Fibonacci
{public static void main(String[] args)    {int
limit=100;int low=0,high=1;System.out.println(
low);while(high<limit){System.out     .println(
high);high=low+high;low=high-low;}              }
}
```

  - This prints the Fibonacci sequence for values < 100 ??
  - In Eclipse: *Ctrl-shift-F* to autoformat the file

Controlling Program Flow

Università di Roma Tor Vergata

- Good programming style
  - Consistent indentation / whitespaces

```
public class Fibonacci {
   public static void main(String[] args) {
      int limit = 100;
      int low = 0, high = 1;
      System.out.println(low);
      while (high < limit) {
         System.out.println(high);
         high = low + high;
         low = high - low;
      }
   }
}
```

Controlling Program Flow

Università di Roma Tor Vergata

- Good programming style
  - Avoiding of duplication / redundancy

```
if (number <= 10) {
    ...
} else if (value > 1.0) {
    ...
} else if (number > 10 && value <= 1.0) {
    ...
}
```

Controlling Program Flow

- Good programming style
  - Avoiding of duplication / redundancy

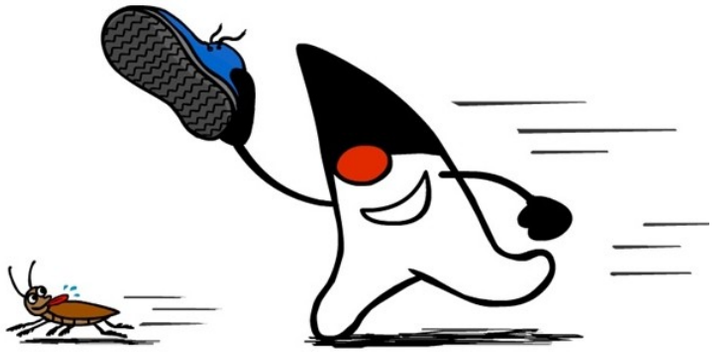
```
if (number <= 10) {
    ...
} else if (value > 1.0) {
    ...
} else {
    ...
}
```

  - DRY (Don't Repeat Yourself), and DIE (Duplication is Evil)
  - Keep your code simple!

Controlling Program Flow

Università di Roma Tor Vergata

- Several types of errors
  - Compilation errors
    - Detected by compiler
    - Mostly due to the violation of syntax rules
  - Execution errors
    - Detected by interpreter while running the program
    - Different sources
  - Wrong results
    - Detected while running the program / testing
    - Mostly logical errors (errors in design of program)
- Testing is crucial!



debugging