Java Web开发基础

第2讲：Java语言基础

主讲人：康育哲

本讲内容

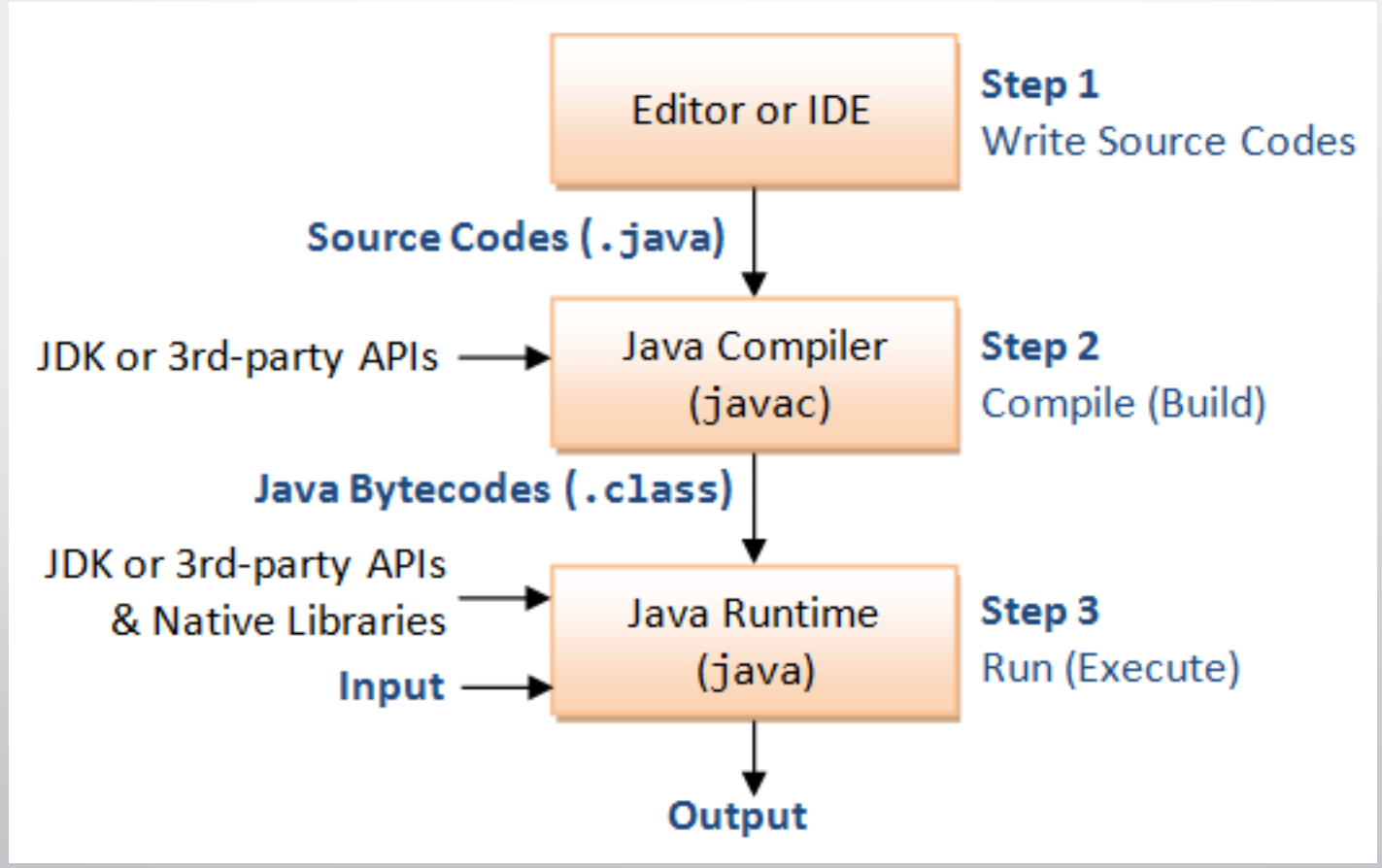
- 回顾与巩固
- 数据类型（Data Types）
- 运算符（Operators）
- 字符串（String）
- 逻辑流程控制
- 基本编程技巧
- 输入/输出（Input/Output）
- 方法（Method）
- 命令行参数（Command-Line Arguments）

回顾与巩固

- Java编译流程
- 注释
- 语句&代码块
- 留白&代码格式
- 变量&常量
- 命名
- 运算符
- 表达式

回顾与巩固

- Java编译流程



回顾与巩固

- 注释
  - 单行注释：// ...
  - 块注释：/* ... */
- 语句：以;结束
- 代码块：{...}

回顾与巩固

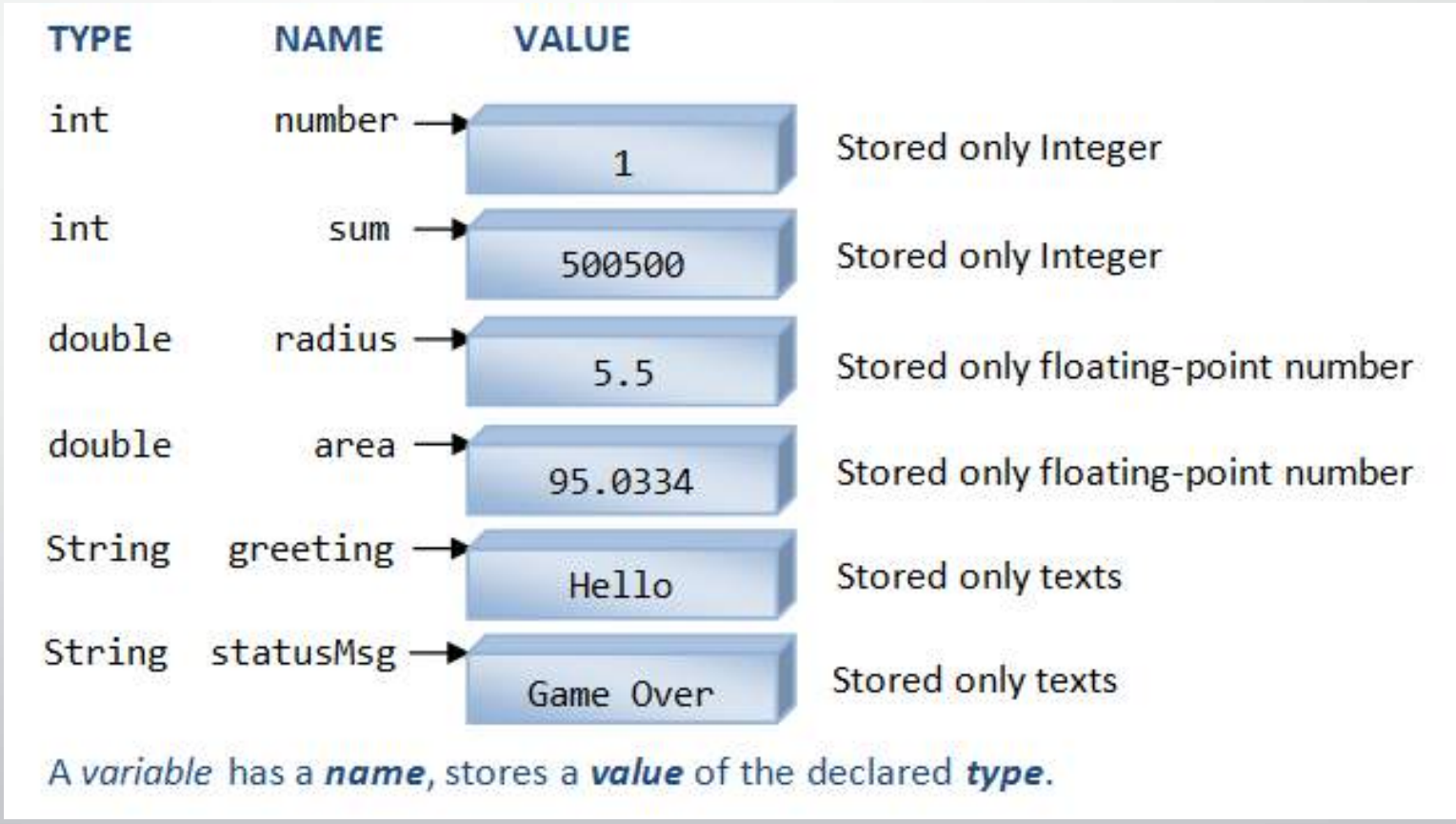
- 留白&代码格式

```
public class Hello{publ
```

```
/*
 * Recommended Java programming style
 */
public class ClassName {      // Place the beginning brace at the end of the current line
   public static void main(String[] args) {  // Indent the body by an extra 3 or 4 spaces for each level

      // Use empty line liberally to improve readability
      // Sequential statements
      statement;
      statement;

      // Conditional statement
      if (test) {
         statements;
      } else {
         statements;
      }

      // loop
      init;
      while (test) {
         statements;
         update;
      }
   }
}   // ending brace aligned with the start of the statement
```

回顾与巩固

- 变量
  - 名称
  - 类型
  - 值
- 常量
  - 关键字：final
  - 赋值一次，不可改变



回顾与巩固

- 变量和常量的命名
  - 命名规则
    - 字符允许范围：a-z、A-Z、0-9、_、$
    - 首字符允许范围：a-z、A-Z、_（$为系统变量的保留首字符）
    - 不允许与Java关键字同名：class、int、if、else、switch、for、while、true、false、null等
    - 大小写敏感
  - 命名规范
    - 驼峰式（camel-case）
      - 名词
      - 多个单词组成，连续排列
      - 首单词首字母小写，后续单词首字母大写
    - 示例： fontSize, roomNumber, xMax, yMin, xTopLeft

回顾与巩固

- 变量和常量的命名
  - 命名技巧
    - 选择意义清晰的名称：e.g. numberOfStudents、result等
    - 不要选择无意义的名称：e.g. xyz、n1等
    - 避免单个字母的名称（循环变量除外）
    - 用复数单词表示复数变量：e.g. Student[] students = new Student[10];

回顾与巩固

- 运算符
  - 赋值运算符：=、+=、-=、*=、/=、%=、++、--
  - 算术运算符：+、-、*、/、%
  - 逻辑运算符：&&、||、!
  - 强制转换运算符：(*type*)
  - 优先级

回顾与巩固

- 表达式
  - 一组运算符（operator）和运算数（operand）的组合，能够得到特定类型的单一结果
  - 示例
  - 表达式vs语句
  - 特殊：赋值表达式

```
1 + 2 * 3              // evaluated to int 7

int sum, number;
sum + number           // evaluated to an int value

double principal, interestRate;
principal * (1 + interestRate)  // Evaluated to a double value
```

数据类型

- 内置类型（built-in types）
- 字符串
- 数据类型的选择
- 字面值（literals）

本讲内容

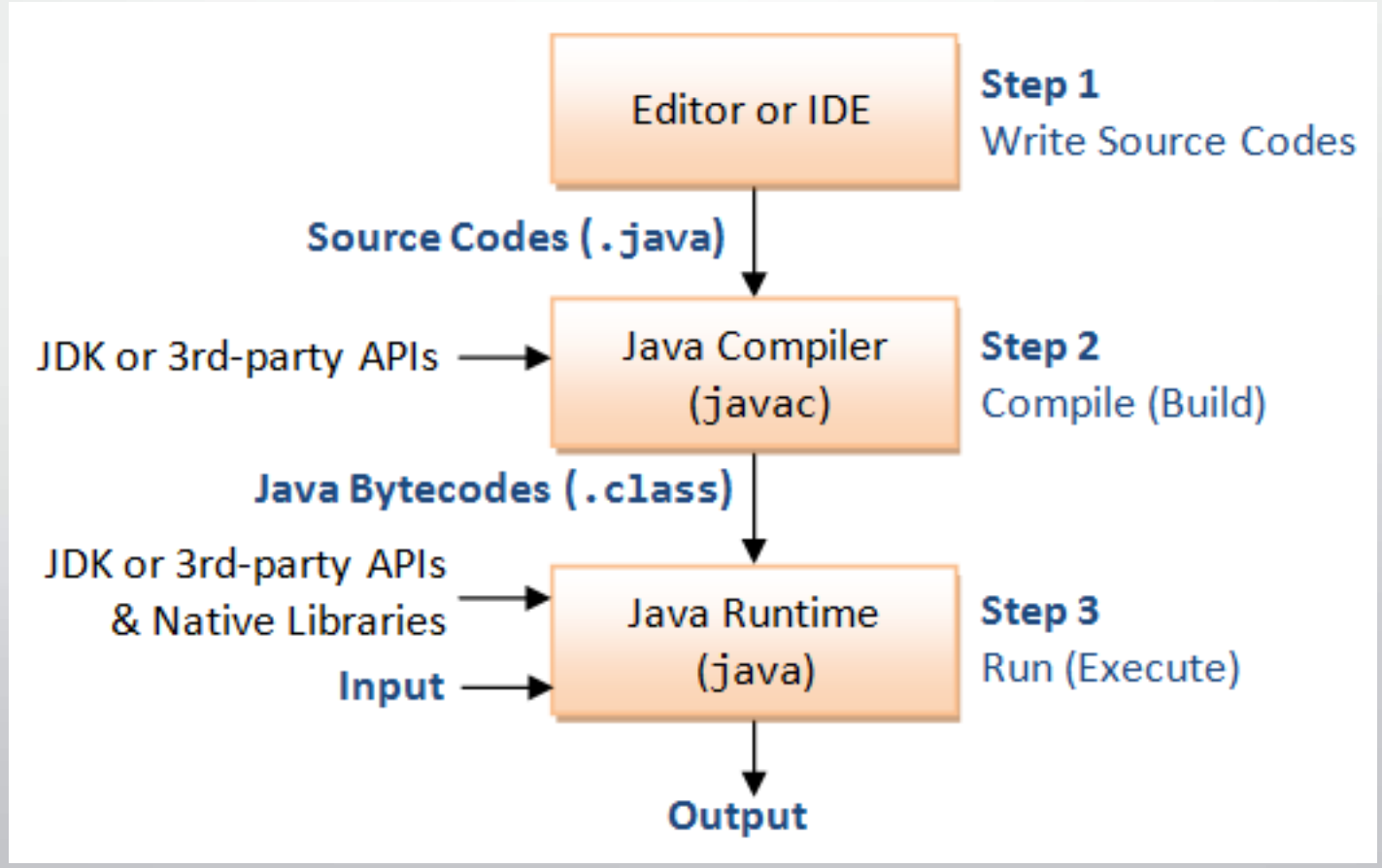
- 回顾与巩固
- 数据类型（Data Types）
- 运算符（Operators）
- 字符串（String）
- 逻辑流程控制
- 基本编程技巧
- 输入/输出（Input/Output）
- 数组（Array）
- 方法（Method）
- 命令行参数（Command-Line Arguments）
- *位运算

回顾与巩固

- Java编译流程
- 注释
- 语句&代码块
- 留白&代码格式
- 变量&常量
- 命名
- 运算符
- 表达式

回顾与巩固

- Java编译流程



回顾与巩固

- 注释
  - 单行注释：// ...
  - 块注释：/* ... */
- 语句：以;结束
- 代码块：{...}

回顾与巩固

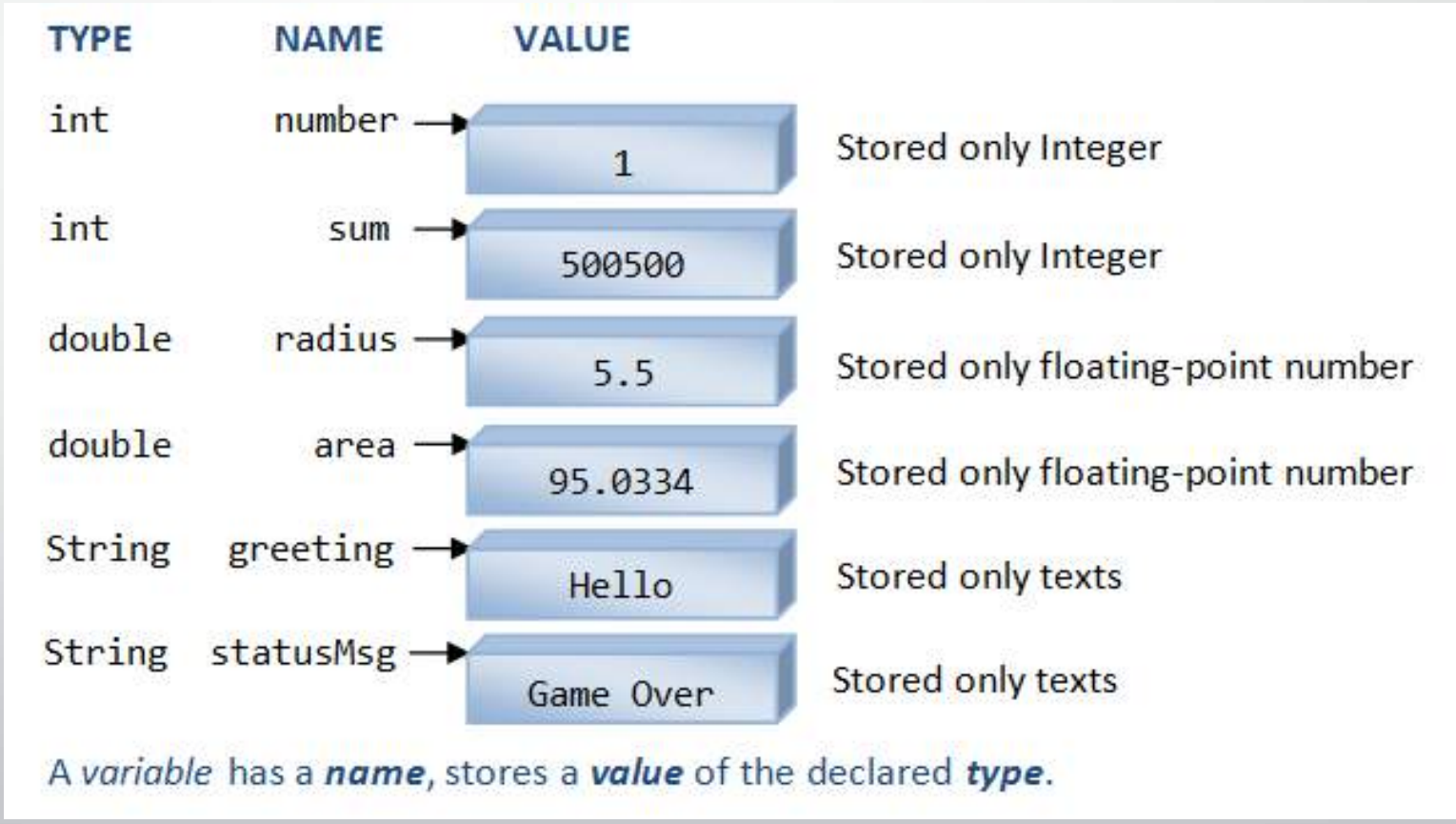
- 留白&代码格式

```
public class Hello{publ
```

```
/*
 * Recommended Java programming style
 */
public class ClassName {      // Place the beginning brace at the end of the current line
   public static void main(String[] args) {  // Indent the body by an extra 3 or 4 spaces for each level

      // Use empty line liberally to improve readability
      // Sequential statements
      statement;
      statement;

      // Conditional statement
      if (test) {
         statements;
      } else {
         statements;
      }

      // loop
      init;
      while (test) {
         statements;
         update;
      }
   }
}   // ending brace aligned with the start of the statement
```

回顾与巩固

- 变量
  - 名称
  - 类型
  - 值
- 常量
  - 关键字：final
  - 赋值一次，不可改变



回顾与巩固

- 变量和常量的命名
  - 命名规则
    - 字符允许范围：a-z、A-Z、0-9、_、$
    - 首字符允许范围：a-z、A-Z、_（$为系统变量的保留首字符）
    - 不允许与Java关键字同名：class、int、if、else、switch、for、while、true、false、null等
    - 大小写敏感
  - 命名规范
    - 驼峰式（camel-case）
      - 名词
      - 多个单词组成，连续排列
      - 首单词首字母小写，后续单词首字母大写
    - 示例： fontSize, roomNumber, xMax, yMin, xTopLeft

回顾与巩固

- 变量和常量的命名
  - 命名技巧
    - 选择意义清晰的名称：e.g. numberOfStudents、result等
    - 不要选择无意义的名称：e.g. xyz、n1等
    - 避免单个字母的名称（循环变量除外）
    - 用复数单词表示复数变量：e.g. Student[] students = new Student[10];

回顾与巩固

- 运算符
  - 赋值运算符：=、+=、-=、*=、/=、%=、++、--
  - 算术运算符：+、-、*、/、%
  - 逻辑运算符：&&、||、!
  - 强制转换运算符：(*type*)
  - 优先级

回顾与巩固

- 表达式
  - 一组运算符（operator）和运算数（operand）的组合，能够得到特定类型的单一结果
  - 示例
  - 表达式vs语句
  - 特殊：赋值表达式

```
1 + 2 * 3              // evaluated to int 7

int sum, number;
sum + number           // evaluated to an int value

double principal, interestRate;
principal * (1 + interestRate)  // Evaluated to a double value
```

数据类型

- 内置类型（built-in types）
- 字符串
- 数据类型的选择
- 字面值（literals）

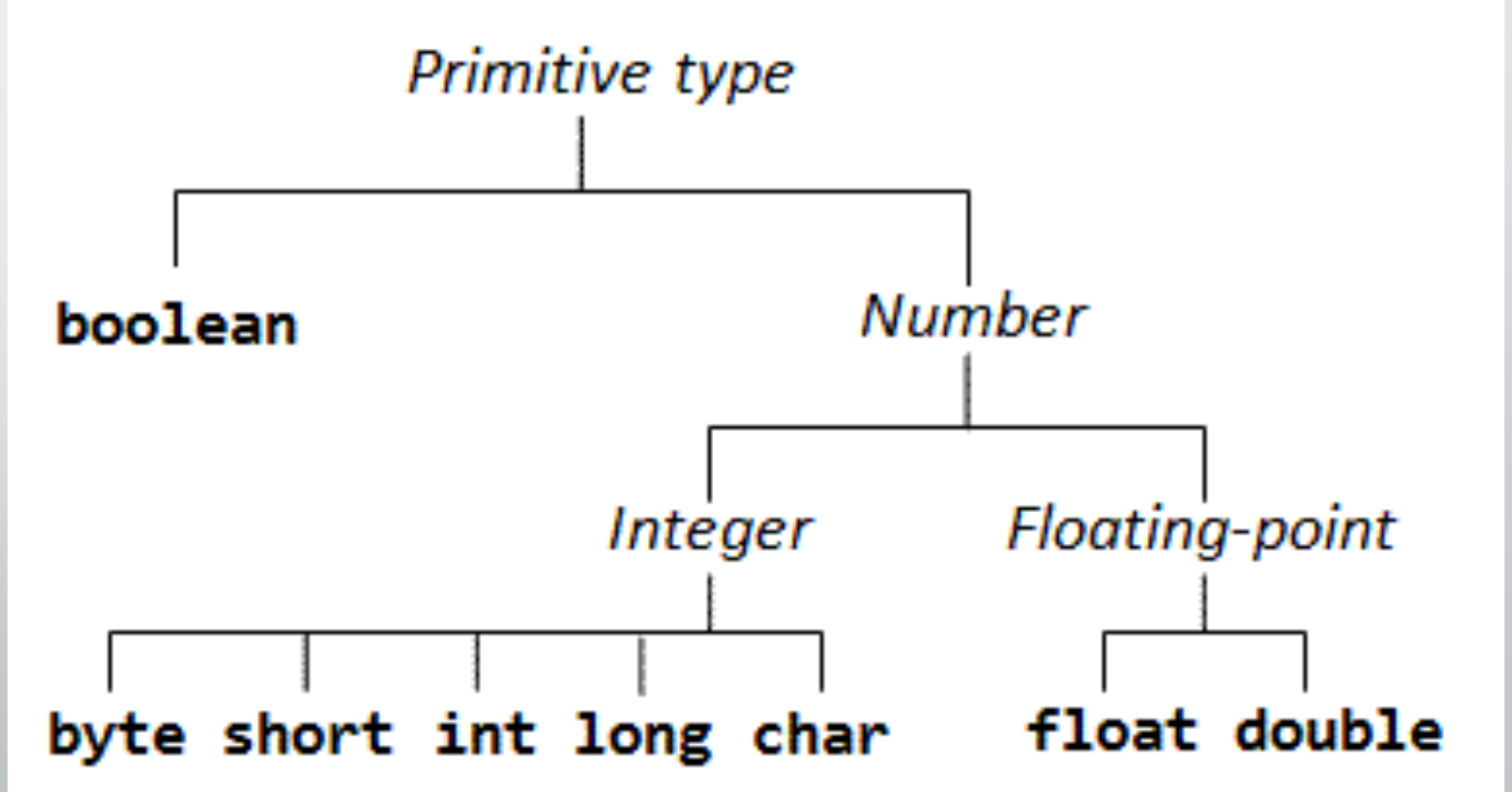
数据类型

- 内置类型

TYPE	DESCRIPTION	
byte	Integer	8-bit signed integer The range is `[-2^7, 2^7-1] = [-128, 127]`
short		16-bit signed integer The range is `[-2^15, 2^15-1] = [-32768, 32767]`
int		32-bit signed integer The range is `[-2^31, 2^31-1] = [-2147483648, 2147483647]` (≈9 digits)
long		64-bit signed integer The range is `[-2^63, 2^63-1] = [-9223372036854775808, +9223372036854775807]` (≈19 digits)
float	Floating-Point Number	32-bit single precision floating-point number (≈6-7 significant decimal digits, in the range of `±[≈10^-45, ≈10^38]`)
double		64-bit double precision floating-point number (≈14-15 significant decimal digits, in the range of `±[≈10^-324, ≈10^308]`)
char	Character Represented in 16-bit Unicode `'\u0000'` to `'\uFFFF'`. Can be treated as 16-bit unsigned integers in the range of `[0, 65535]` in arithmetic operations.	
boolean	Binary Takes a value of either `true` or `false`. The size of `boolean` is not defined in the Java specification, but requires at least one bit.	

数据类型

- 内置类型
  - 与类的关系



数据类型

- 内置类型
  - 获取值域和位长
    - *type*.MIN_VALUE
    - *type*.MAX_VALUE
    - *type*.SIZE

数据类型

- 字符串
  - 由一组字符（char）组成的数据，代表文本
  - 由于Java字符是Unicode编码，所以字符串默认也是Unicode编码
  - 字符串属于类，而不是内置类型

数据类型

- 数据类型的选择
  - 通常情况下，选择int表示整型数据，选择double表示浮点型数据
  - 除非有足够的理由，否则尽量不用byte、short、long、float
  - 循环变量或计数变量使用int
  - 数据含有小数部分则使用double，没有小数部分则使用int

数据类型

- 字面值
  - 又称字面常量，是一种特殊的常量，直接用其来值表示
  - 例：123, -456, 3.14, -1.2e3, 'a', "Hello”
  - 可用于赋值或参与表达式的运算

数据类型

- 字面值
  - 整型

```
int number = -123;
int sum = 1234567890;      // This value is within the range of int
int bigSum = 8234567890;   // ERROR: this value is outside the range of int
```

```
long bigNumber = 1234567890123L;   // Suffix 'L' needed
long sum = 123;                    // int 123 auto-casts to long 123L
```

    - int：正常形式的八进制、十进制、十六进制整数
    - long：后缀为L或l的整数
    - byte & short：没有特殊后缀，但只能使用值域范围内的值

```
byte smallNumber = 12345;      // ERROR: this value is outside the range of byte.
byte smallNumber = 123;        // This is within the range of byte
short midSizeNumber = -12345;
```

数据类型

- 字面值
  - 布尔型：true或false
  - 浮点型
    - double：后缀为D或d或者不带后缀的实数
    - float：后缀为F或f的实数，必须带后缀

```
float average = 55.66;       // Error! RHS is a double. Need suffix 'f' for float.
float average = 55.66f;
```

数据类型

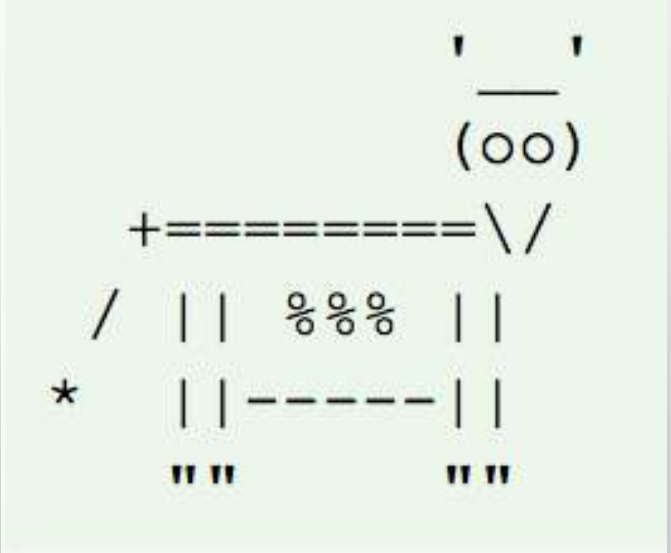
- 字面值
  - 字符：字符型或整型数据
  - 转义字符（escape sequence）

```
char letter = 'a';                  // Same as 97
char anotherLetter = 98;            // Same as the letter 'b'
System.out.println(letter);         // 'a' printed
System.out.println(anotherLetter); // 'b' printed instead of the number
anotherLetter += 2;                 // 100 or 'd'
System.out.println(anotherLetter); // 'd' printed
```

Escape Sequence	Description	Unicode in Hex (Decimal)
\n	Newline (or Line-feed)	000AH (10D)
\r	Carriage-return	000DH (13D)
\t	Tab	0009H (9D)
\"	Double-quote	0022H (34D)
\'	Single-quote	0027H (39D)
\\	Back-slash	005CH (92D)
\u*hhhh*	Unicode number *hhhh* (in hex), e.g., \u000a is newline, \u60a8 is 您, \u597d is 好	*hhhh*H

数据类型

- 小练习（字符串常量和转义字符）
  - 请在控制台打印出下面这只小羊。



运算符

- 算术运算符
- 类型混合运算
- 溢出
- 类型转换运算符
- 复合运算符
- 自增/自减运算符
- 逻辑运算符
- *位运算符

运算符

- 算术运算符

Operator	Description	Usage	Examples
*	Multiplication	*expr1* * *expr2*	2 * 3 → 6 3.3 * 1.0 → 3.3
/	Division	*expr1* / *expr2*	1 / 2 → 0 1.0 / 2.0 → 0.5
%	Remainder (Modulus)	*expr1* % *expr2*	5 % 2 → 1 -5 % 2 → -1 5.5 % 2.2 → 1.1
+	Addition (or unary positive)	*expr1* + *expr2* +expr	1 + 2 → 3 1.1 + 2.2 → 3.3
-	Subtraction (or unary negate)	*expr1* - *expr2* -expr	1 - 2 → -1 1.1 - 2.2 → -1.1

运算符

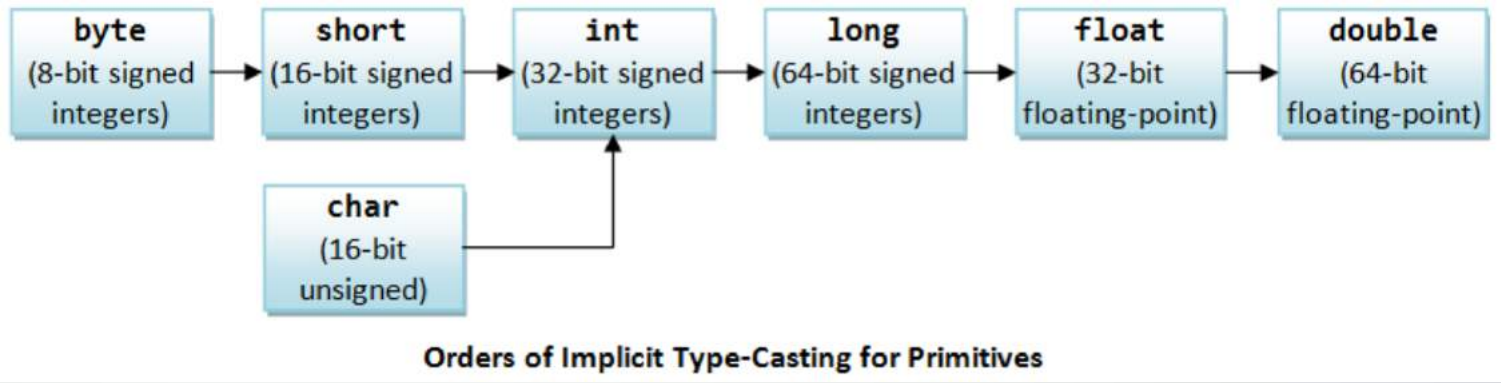
- 类型混合运算
  - 同类型运算数的运算结果还是该类型
    - 两个byte型数据的运算结果被提升为int型
    - 对char、byte、short做相反数运算（-），结果被提升为int型
    - 注意点：整型数据做除法结果还是整型，小数部分会被无条件舍弃
  - 不同类型运算数的运算结果被提升为高精度类型
  - 负数和实数求余：辗转相减法

```
-5 % 2 ⇒ -3 % 2 ⇒ -1
5.5 % 2.2 ⇒ 3.3 % 2.2 ⇒ 1.1
```

运算符

- 溢出
  - 运算结果超出数据类型的值域
  - Java编译器和JVM均不检查溢出
  - 上溢（overflow）
  - 下溢（underflow）

运算符



Orders of Implicit Type-Casting for Primitives

- 类型转换
  - 隐式转换
    - 低精度数据赋给高精度类型的变量
    - 不需要类型转换运算符
  - 显式转换
    - 高精度数据赋给低精度类型的变量
    - 必须要加类型转换运算符(*type*)

运算符

- 复合运算符

Operation	Description	Usage	Example
=	Assignment Assign the value of the LHS to the variable at the RHS	*var* = *expr*	x = 5;
+=	Compound addition and assignment	*var* += *expr* same as *var* = *var* + *expr*	x += 5; same as x = x + 5
-=	Compound subtraction and assignment	*var* -= *expr* same as *var* = *var* - *expr*	x -= 5; same as x = x - 5
*=	Compound multiplication and assignment	*var* *= *expr* same as *var* = *var* * *expr*	x *= 5; same as x = x * 5
/=	Compound division and assignment	*var* /= *expr* same as *var* = *var* / *expr*	x /= 5; same as x = x / 5
%=	Compound remainder (modulus) and assignment	*var* %= *expr* same as *var* = *var* % *expr*	x %= 5; same as x = x % 5

运算符

- 自增/自减运算符

Operator	Description	Example
`++var`	Pre-Increment Increment *var*, then use the new value of *var*	`y = ++x;` same as `x=x+1; y=x;`
`var++`	Post-Increment Use the old value of *var*, then increment *var*	`y = x++;` `same as oldX=x; x=x+1; y=oldX;`
`--var`	Pre-Decrement	`y = --x;` `same as x=x-1; y=x;`
`var--`	Post-Decrement	`y = x--;` `same as oldX=x; x=x-1; y=oldX;`

运算符

- 逻辑运算符
  - 单一逻辑

Operator	Description	Usage	Example (x=5, y=8)
==	Equal to	*expr1* == *expr2*	(x == y) → false
!=	Not Equal to	*expr1* != *expr2*	(x != y) → true
>	Greater than	*expr1* > *expr2*	(x > y) → false
>=	Greater than or equal to	*expr1* >= *expr2*	(x >= 5) → true
<	Less than	*expr1* < *expr2*	(y < 8) → false
<=	Less than or equal to	*expr1* >= *expr2*	(y <= 8) → true

运算符

- 逻辑运算符
  - 复合逻辑

Operator	Description	Usage
!	Logical NOT	`!booleanExpr`
^	Logical XOR	`booleanExpr1 ^ booleanExpr2`
&&	Logical AND	`booleanExpr1 && booleanExpr2`
||	Logical OR	`booleanExpr1 || booleanExpr2`

运算符

- 逻辑运算符
  - 复合逻辑：真值表（truth table）

AND（&&）	true	false
true	true	false
false	false	false

OR（||）	true	false
true	true	true
false	true	false

NOT（!）	true	false
Result	false	true

XOR（^）	true	false
true	false	true
false	true	false

字符串

- 拼接操作符：+
  - 字符串之间拼接
  - 字符串与其他类型数据之间拼接

```
"Hello" + "world" → "Helloworld"
"Hi" + ", " + "world" + "!" → "Hi, world!"
```

```
"The number is " + 5 → "The number is " + "5" → "The number is 5"
"The average is " + average + "!" (suppose average=5.5) → "The average is " + "5.5" + "!" → "The average is 5.5!"
"How about " + a + b (suppose a=1, b=1) → "How about 11"
```

字符串

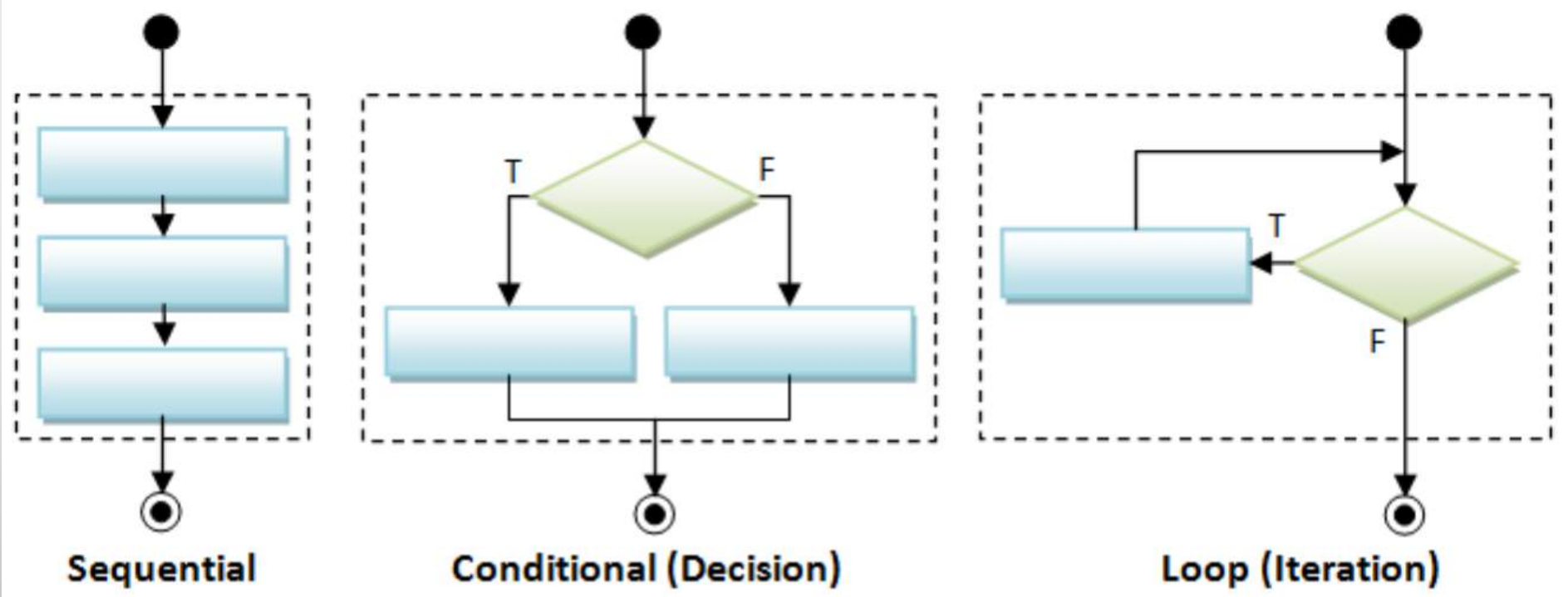
- 字符串基本运算
  - length()：取长度
  - charAt(int index)：取字符
  - equals(String anotherStr)：判等
    - equalsIgnoreCase()：忽略大小写的判等

字符串

- 字符串与内置类型的互转
  - 字符串转内置类型：parse系列方法
  - 内置类型转字符串：toString系列方法

逻辑流程控制

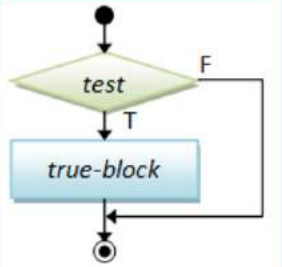
- 顺序
- 分支
- 循环

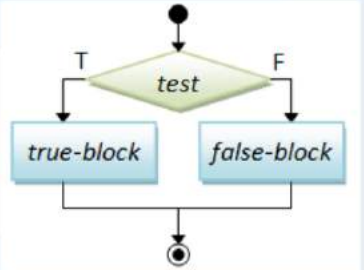


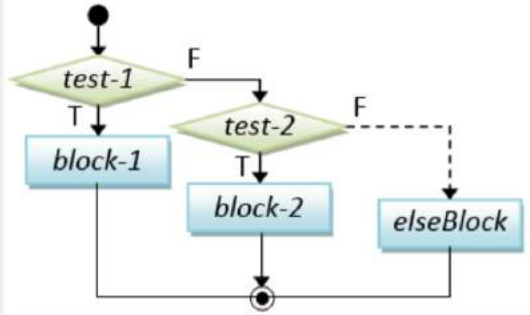
逻辑流程控制

- 分支
  - if-else系
    - if
    - if-else
    - if-else if-else

Syntax	Example	Flowchart

// if-then
if (*booleanExpression*) {
  true-block ;
}
	if (mark >= 50) {
	  System.out.println("Congratulation!");
	  System.out.println("Keep it up!");
	}
		

// if-then-else
if (*booleanExpression*) {
  true-block ;
} else {
  false-block ;
}
	if (mark >= 50) {
	  System.out.println("Congratulation!");
	  System.out.println("Keep it up!");
	} else {
	  System.out.println("Try Harder!");
	}
		

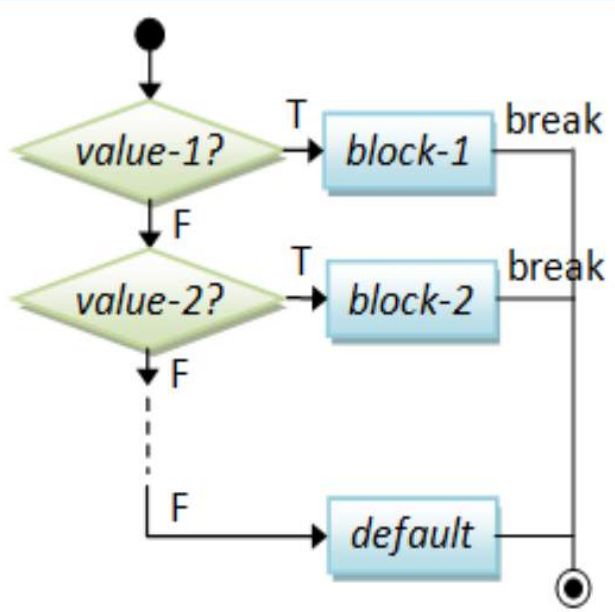
// nested-if
if (*booleanExpr-1*) {
  block-1 ;
} else if (*booleanExpr-2*) {
  block-2 ;
} else if (*booleanExpr-3*) {
  block-3 ;
} else if (*booleanExpr-4*) {
  ......
} else {
  elseBlock ;
}
	if (mark >= 80) {
	  System.out.println("A");
	} else if (mark >= 70) {
	  System.out.println("B");
	} else if (mark >= 60) {
	  System.out.println("C");
	} else if (mark >= 50) {
	  System.out.println("D");
	} else {
	  System.out.println("F");
	}
		

逻辑流程控制

- 分支
  - switch-case系

```
// switch-case-default
switch ( selector ) {
   case value-1:
      block-1; break;
   case value-2:
      block-2; break;
   case value-3:
      block-3; break;
   ......
   case value-n:
      block-n; break;
   default:
      default-block;
}
```

```
char oper; int num1, num2, result;
......
switch (oper) {
   case '+':
      result = num1 + num2; break;
   case '-':
      result = num1 - num2; break;
   case '*':
      result = num1 * num2; break;
   case '/':
      result = num1 / num2; break;
   default:
      System.err.println("Unknown operator);
}
```



逻辑流程控制

- 分支
  - ?系

Syntax	Example

booleanExpr **?** *trueExpr* **:** *falseExpr*	`System.out.println((mark >= 50) ? "PASS" : "FAIL");`
	`max = (a > b) ? a : b;  // RHS returns a or b`
	`abs = (a > 0) ? a : -a;  // RHS returns a or -a`

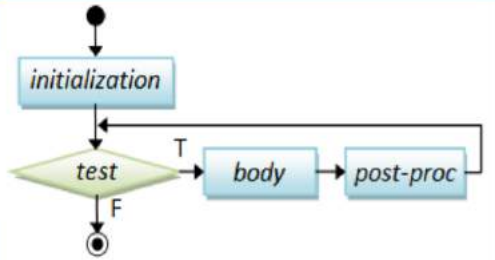
逻辑流程控制

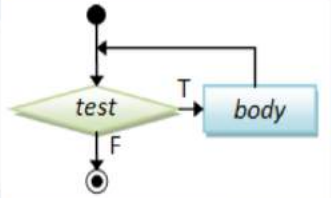
- 小练习（分支）
  - 逢七必过：从1打印到100，出现7的数字或7的倍数，一律打印成PASS。

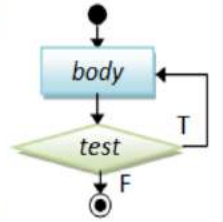
逻辑流程控制

- 循环
  - while系
    - while
    - do-while
  - for系
    - for(;;)
    - for(:)

Syntax	Example	Flowchart

`// for-loop`
`for (initialization; test; post-processing) {`
`  body;`
`}`
	`// Sum from 1 to 1000`
	`int sum = 0;`
	`for (int number = 1; number <= 1000; ++number) {`
	`  sum += number;`
	`}`
		

`// while-do loop`
`while ( test ) {`
`  body;`
`}`
	`int sum = 0, number = 1;`
	`while (number <= 1000) {`
	`  sum += number;`
	`  ++number;`
	`}`
		

`// do-while loop`
`do {`
`  body;`
`}`
`while ( test ) ;`
	`int sum = 0, number = 1;`
	`do {`
	`  sum += number;`
	`  ++number;`
	`} while (number <= 1000);`
		

逻辑流程控制

- 循环
  - while与do-while的区别：do-while至少执行一次
  - for(:)存在的意义：便于轮询数据集合
  - 跳出循环：break
  - 跳入下一次循环：continue
  - 循环嵌套

逻辑流程控制

- 循环
  - 循环嵌套示例

```
1  /*
2   * Print a square pattern
3   */
4  public class PrintSquarePattern {    // Save as "PrintSaurePattern.java"
5     public static void main(String[] args) {
6        int size = 8;
7        for (int row = 1; row <= size; ++row) {     // Outer loop to print all the rows
8           for (int col = 1; col <= size; ++col) {  // Inner loop to print all the columns of each row
9              System.out.print("# ");
10          }
11          System.out.println();   // A row ended, bring the cursor to the next line
12       }
13    }
14 }
```

基本编程技巧

- 排错
- 调试
- 测试

基本编程技巧

- 排错
  - 编译错误：非常容易发现，容易修复，危害小
  - 运行时错误：容易发现，容易修复，编码得当可以把危害降到最低
  - 逻辑错误：不容易发现，修复成本较高，危害较大，避免这类错误不仅需要规范的编码，而且需要仔细的设计和严格测试

基本编程技巧

- 调试
  - 插入print语句：破坏代码结构
  - 使用UI调试器：不用修改代码，但对于性能问题查不出来
  - 使用profiler等高级工具：不用跟踪代码，便于分析性能问题

输入/输出

- 控制台I/O
- 文件I/O

输入/输出

- 控制台I/O
  - 格式化输出：printf
    - 常见参数格式
      - %*n*d：整型参数，n为所占宽度（可选）
      - %*n*s：字符串参数，n为所占宽度（可选）
      - %*a*.*b*f：浮点型参数，a为所占宽度（可选），b为精度（可选）
  - 读取输入：Scanner

输入/输出

- 文件I/O
  - 文件输入Scanner
  - 文件输出Formatter
  - 基本异常处理

方法（函数）

- 本质
- 用途
- 调用逻辑
- 声明格式
- 命名规范
- 参数
- 重载

方法（函数）

- 本质：代码块
- 用途：
  - 细分问题
  - 代码复用
  - 软件复用

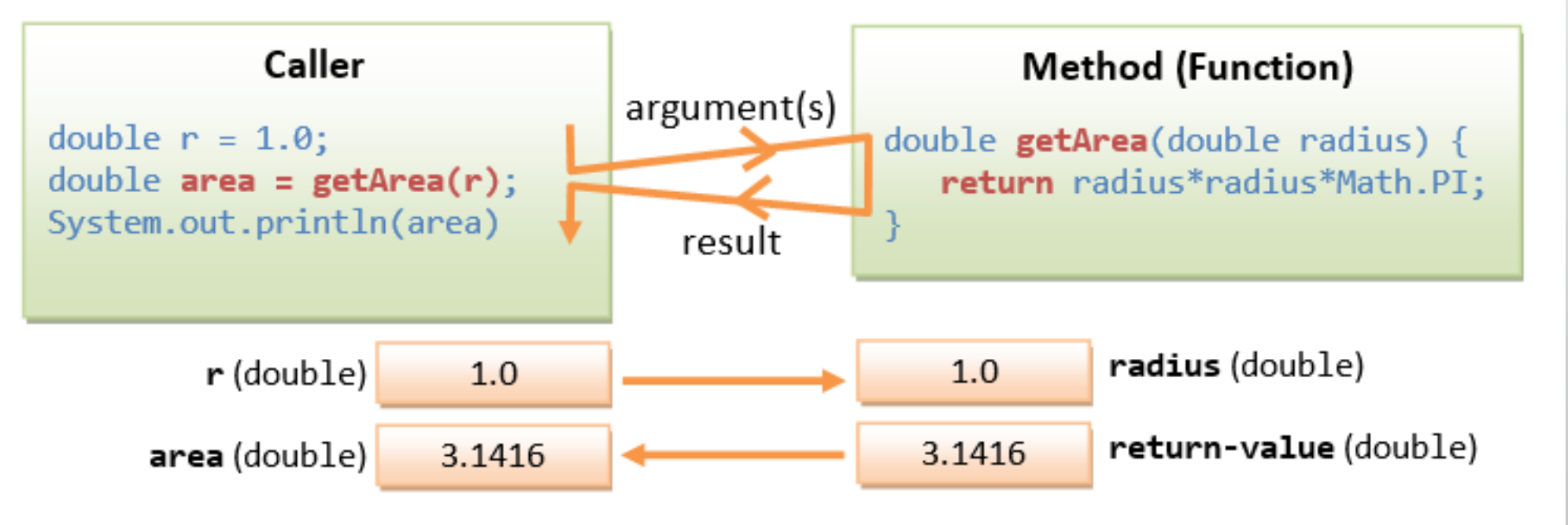
方法（函数）

- 调用逻辑

```
1  public class EgMethodGetArea {
2     // The entry main method
3     public static void main(String[] args) {
4        double r = 1.1, area, area2;
5        // Call (Invoke) method getArea()
6        area = getArea(r);
7        System.out.println("area is " + area);
8        // Call method getArea() again
9        area2 = getArea(2.2);
10       System.out.println("area 2 is " + area2);
11       // Call method getArea() one more time
12       System.out.println("area 3 is " + getArea(3.3));
13    }
14
15    // Method getArea() Eefinition.
16    // Compute and return the area (in double) of circle given its radius (in double).
17    public static double getArea(double radius) {
18       return radius * radius * Math.PI;
19    }
20 }
```

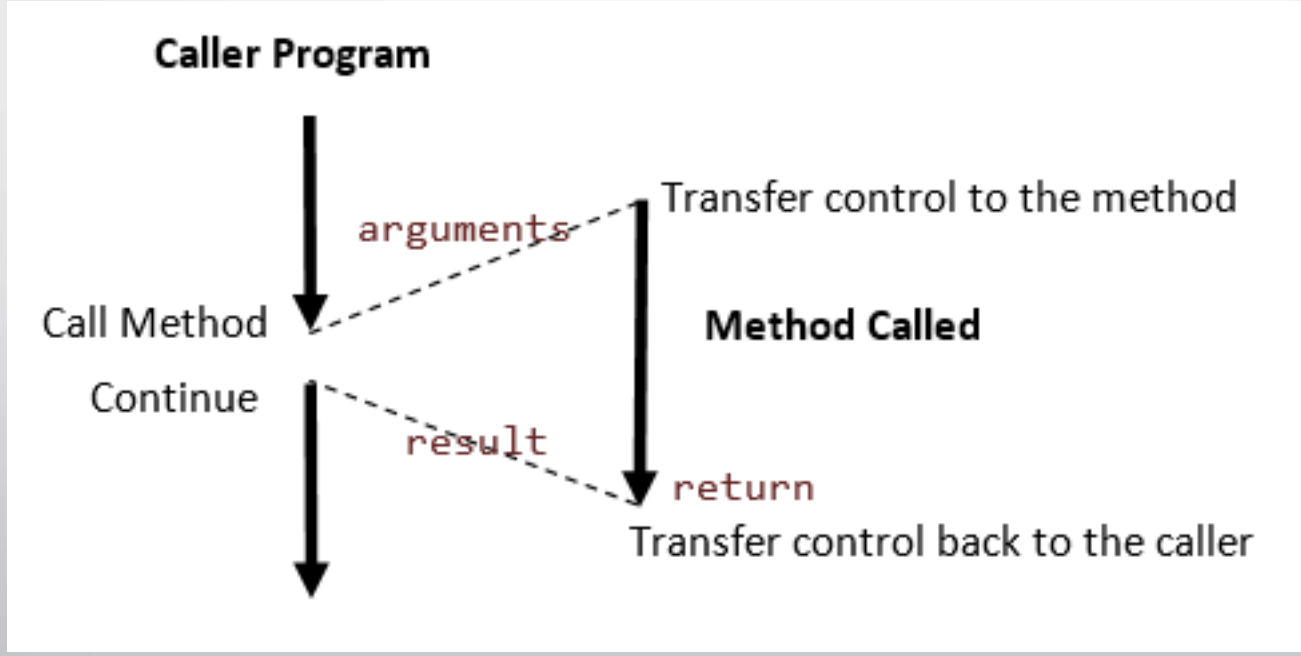
方法（函数）

- 调用逻辑


Caller
double r = 1.0;
double area = getArea(r);
System.out.println(area)
argument(s)
result
Method (Function)
double getArea(double radius) {
return radius*radius*Math.PI;
}
r (double)
1.0
1.0
radius (double)
area (double)
3.1416
3.1416
return-value (double)

方法（函数）

- 调用逻辑


Caller Program
arguments
Transfer control to the method
Call Method
Method Called
Continue
result
return
Transfer control back to the caller

方法（函数）

- 声明格式

```
public static returnValueType methodName ( arg-1-type arg-1, arg-2-type arg-2,... ) {
    body ;
}
```

  - 返回值：可以为某数据类型或空
  - 参数：可以是固定参数或可变参数
- 命名规范
  - 动词或动词词组，如getArea()、setPrice()、remove()等
  - 驼峰式命名

方法（函数）

- 参数
  - 形式参数：函数签名中的参数
    - 如：double getArea(double radius)
  - 实际参数：函数调用时传入的参数
    - 如：area1=getArea(r)

方法（函数）

- 参数
  - 固定参数：参数数量固定
  - 可变参数：参数数量不固定

```java
public class VarargsTest {
    // A method which takes a variable number of arguments (varargs)
    public static void doSomething(String... strs) {
        System.out.print("Arguments are: ");
        for (String str : strs) {
            System.out.print(str + ", ");
        }
        System.out.println();
    }

    // A method which takes exactly two arguments
    public static void doSomething(String s1, String s2) {
        System.out.println("Overloaded version with 2 args: " + s1 + ", " + s2);
    }

    // Cannot overload with this method - crash with varargs version
    // public static void doSomething(String[] strs)

    // Test main() method
    // Can also use String... instead of String[]
    public static void main(String... args) {
        doSomething("Hello", "world", "again", "and", "again");
        doSomething("Hello", "world");

        String[] strs = {"apple", "orange"};
        doSomething(strs);  // invoke varargs version
    }
}
```

方法（函数）

- 重载
  - 方法名相同，参数定义不同
  - 编译器能自动识别应该调用的方法版本
  - 参数可以被隐式转换

```
/** Testing Method Overloading */
public class EgMethodOverloading {
   public static void main(String[] args) {
      System.out.println(average(8, 6));      // invoke version 1
      System.out.println(average(8, 6, 9));   // invoke version 2
      System.out.println(average(8.1, 6.1)); // invoke version 3
      System.out.println(average(8, 6.1));
           // int 8 autocast to double 8.0, invoke version 3
      // average(1, 2, 3, 4)  // Compilation Error - no such method
   }
   // Version 1 takes 2 int's
   public static int average(int n1, int n2) {
      System.out.println("version 1");
      return (n1 + n2)/2;  // int
   }
   // Version 2 takes 3 int's
   public static int average(int n1, int n2, int n3) {
      System.out.println("version 2");
      return (n1 + n2 + n3)/3;   // int
   }
   // Version 3 takes 2 doubles
   public static double average(double n1, double n2) {
      System.out.println("version 3");
      return (n1 + n2)/2.0;  // double
   }
}
```

命令行参数

- Java的main方法携带参数args，即为整个程序的命令行参数
- 用法举例

```
java Arithmetic 3 2 +
3+2=5
java Arithmetic 3 2 -
3-2=1
java Arithmetic 3 2 /
3/2=1
```

```
1  public class Arithmetic {
2     public static void main (String[] args) {
3        int operand1, operand2;
4        char theOperator;
5        operand1 = Integer.parseInt(args[0]);  // Convert String to int
6        operand2 = Integer.parseInt(args[1]);
7        theOperator = args[2].charAt(0);       // Consider only 1st character
8        System.out.print(args[0] + args[2] + args[1] + "=");
9        switch(theOperator) {
10          case ('+'):
11             System.out.println(operand1 + operand2); break;
12          case ('-'):
13             System.out.println(operand1 - operand2); break;
14          case ('*'):
15             System.out.println(operand1 * operand2); break;
16          case ('/'):
17             System.out.println(operand1 / operand2); break;
18          default:
19             System.out.printf("%nError: Invalid operator!");
20       }
21    }
22 }
```

命令行参数

- 小练习
  - 位数之和：写程序（Java类命名为SumDigits），计算一个正整数的各位数之和。程序使用方法和输出示例如下：

```
java SumDigits 12345
The sum of digits = 1 + 2 + 3 + 4 + 5 = 15
```