DM537
Object-Oriented Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM537/

COURSE ORGANIZATION

UNIVERSITY OF SOUTHERN **DENMARK**.DK

Course Elements

- Lectures Monday 12-14 (every week)
- Lectures Wednesday 12-14 (every other week from next)

- 4 sections (???):
    - M1: Mathematics-Economy (2nd year)
    - S1: Applied Mathematics / Physics / … (2nd year)
    - S7 & S17: Computer Science (1st year)

- Discussion sections (marked "E" in your schedule)
- Labs (marked "L" in your schedule)

- Exam = practical project in 2 parts

UNIVERSITY OF SOUTHERN DENMARK.DK

Course Goals

- **Write non-trivial computer programs**

- To this end, you will learn
    - how to structure programs into classes
    - to use advanced object-oriented techniques
    - to encapsulate functionality in abstract data types

- Focus on general principles, NOT on the language Java

Practical Issues / Course Material

- Regularly check the course home page:
  - http://imada.sdu.dk/~petersk/DM537/
  - Slides, weekly notes, definite schedule, additional notes

- Reading material:
  - David J. Eck: *Introd. to Programming using Java*, Lulu, 2012.
  - Available as PDF and HTML from:
    http://math.hws.edu/javanotes/

  - Allen B. Downey: *Think Java*, Green Tea Press, 2012.
  - Available as PDF and HTML from:
    http://greenteapress.com/thinkapjava/
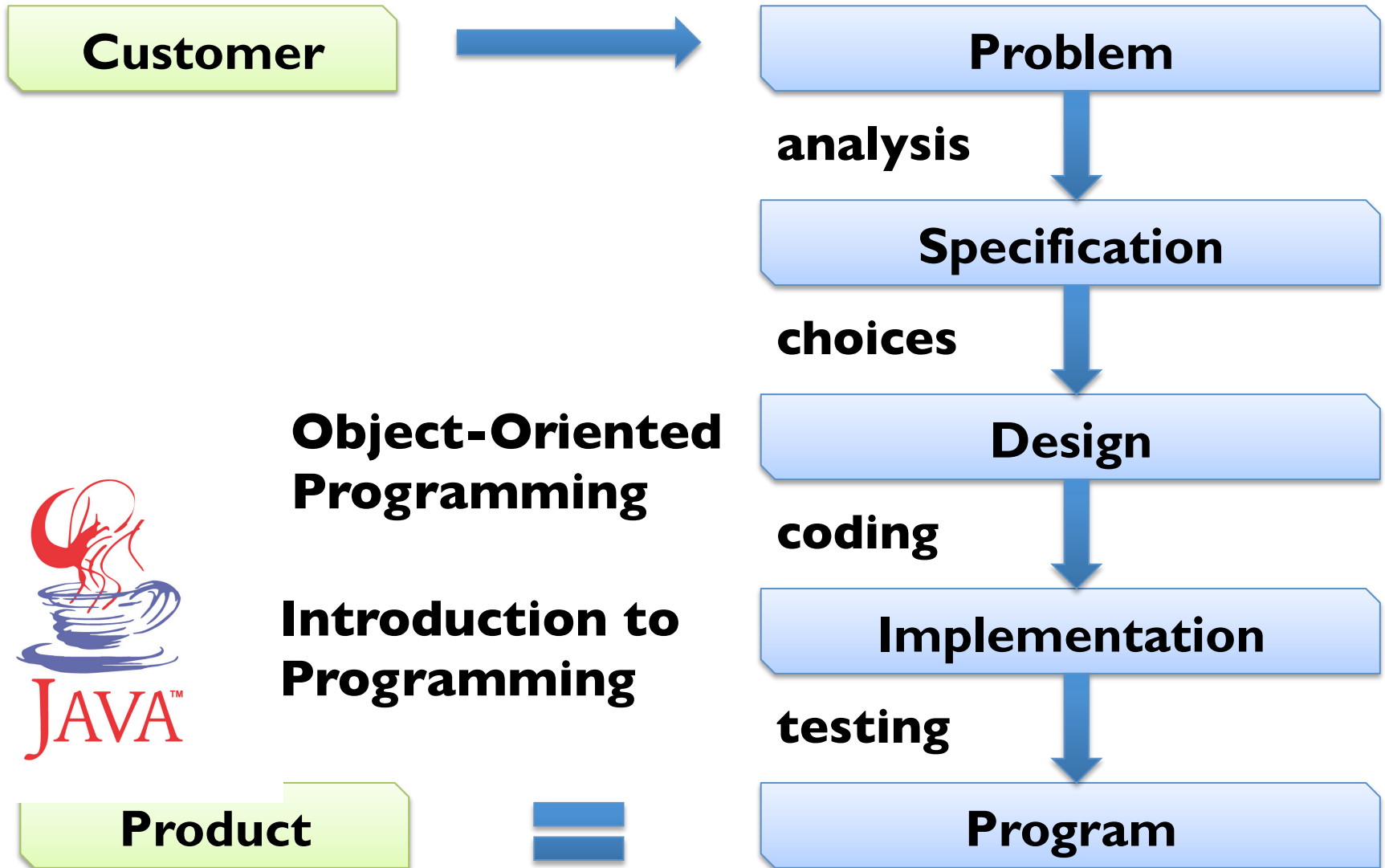
Course Contract 2.0

- I am offering you the following:
    1. I explain all needed concepts (as often as needed)
    2. I try to be available and am always willing to help you
    3. I guide your learning by assigning exercises

- From you I expect the following:
    1. You ask questions, when something is unclear
    2. You contact me (or a TA), when you need help
    3. *You program early and you program often!*

- You and I have the right and duty to call upon the contract!

PROGRAMMING

Programming as Problem Solving

Customer → **Problem**

analysis

Specification

choices

Object-Oriented Programming

Design

coding

Introduction to Programming

Implementation

testing

Product = **Program**

Simple Instructions

- Administrative: import java.util.Scanner;

- Input: s = new Scanner(System.in);

    a = s.nextInt();

    b = s.nextInt();

- Arithmetic operations: c = Math.sqrt(a*a+b*b);
- Output: System.out.println("Result: "+c);

- That is basically ALL a computer can do.

Simple Instructions

```
import java.util.Scanner;


s = new Scanner(System.in);

a = s.nextInt();

b = s.nextInt();

c = Math.sqrt(a*a+b*b);

System.out.println("Result: "+c);
```

Simple Instructions

```java
import java.util.Scanner;
public class Pythagoras {
        public static void main(String[] as) {
                s = new Scanner(System.in);
                a = s.nextInt();
                b = s.nextInt();
                c = Math.sqrt(a*a+b*b);
                System.out.println("Result: "+c);
        } // main
} // Pythagoras
```
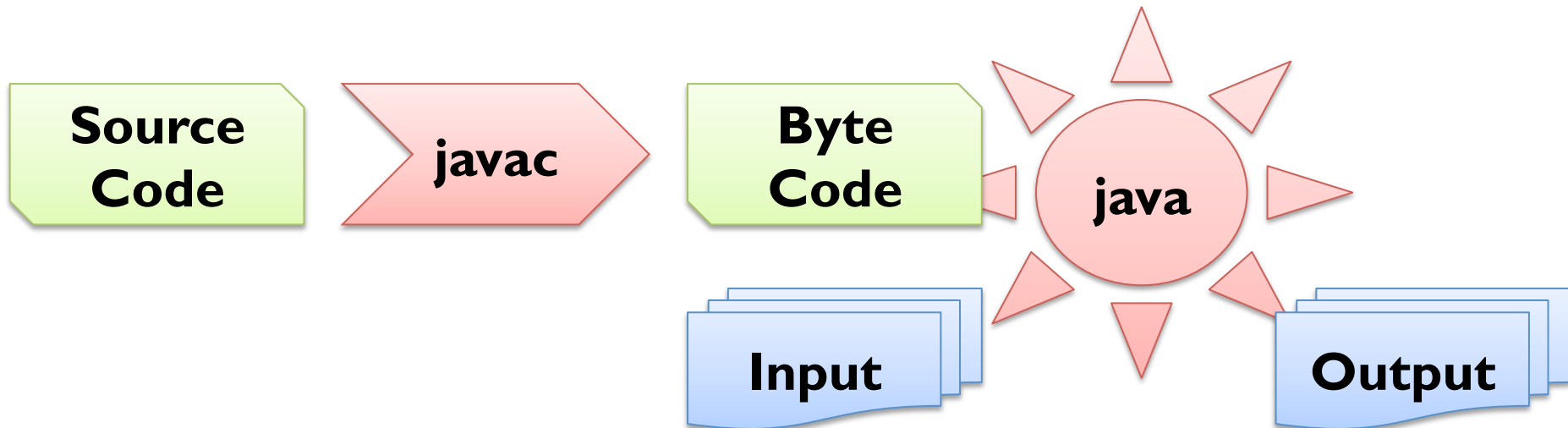
Simple Instructions

```java
import java.util.Scanner;
public class Pythagoras {
        public static void main(String[] as) {
                Scanner s = new Scanner(System.in);
                int a = s.nextInt();
                int b = s.nextInt();
                double c = Math.sqrt(a*a+b*b);
                System.out.println("Result: "+c);
        } // main
} // Pythagoras
```

Combining Instructions

- Sequence: $<instr_1>$; $<instr_2>$; $<instr_3>$;

- Conditional Execution: if (<cond>) {

  $<instr_1>$; $<instr_2>$;

  } else {

  $<instr_3>$; $<instr_4>$; $<instr_5>$;

  }

- Subprograms / Functions: <type> <function>(<argument>) {

  $<instr_1>$; $<instr_2>$;

  }

- Repetition: while (<cond>) {

  $<instr_1>$; $<instr_2>$; $<instr_3>$;

  }

Executing Programs

- Program stored in a file (*source code* file)
- Program is *compiled* to machine-readable code (*byte code*)
- Java *Virtual Machine* (JVM) executes byte code

UNIVERSITY OF SOUTHERN DENMARK.DK

Debugging

- Any reasonably complex program contains errors
- Three types of errors (in Java)

  - Compiler Errors
    - Syntactic Errors   public ssalc HelloWorld {}
    - Type Errors   int a = new Scanner();

  - Runtime Errors   int c = 42 / 0;

  - Semantic Errors   int c = a*a+b*b;

- Debugging is finding out why an error occurred

UNIVERSITY OF SOUTHERN DENMARK.DK

VARIABLES, EXPRESSIONS & STATEMENTS

Values and Types

- Values = basic data objects   42   23.0   "Hello!"
- Types = classes of values   int   double   String

- Types need to be declared
  - `<type> <var>`;   int answer;

- Values can be printed:
  - `System.out.println(<value>);`   `System.out.println(23.0);`

- Values can be compared:
  - `<value> == <value>`   -3 == -3.0

Variables

- variable = name that refers to value of certain type
- program state = mapping from variables to values

- values are *assigned* to variables using "=":
    - `<var> = <value>;`   `answer = 42;`

- the value referred to by a variable can be printed:
    - `System.out.println(<var>);`   `System.out.println(answer);`

- the type of a variable is given by its declaration

Primitive Types

Type	Bits	Range
boolean	1	{true, false}
byte	8	$\{-2^7 = -128, \ldots, 127 = 2^7-1\}$
short	16	$\{-2^{15} = -32768, \ldots, 32767 = 2^{15}-1\}$
char	16	{'a', …,'z', '%', …}
int	32	$\{-2^{31}, \ldots, 2^{31}-1\}$
float	32	1 sign, 23(+1) mantissa, 8 exponent bits
long	64	$\{-2^{63}, \ldots, 2^{63}-1\}$
double	64	1 sign, 52(+1) mantissa, 11 exponent bits

Reference Types

- references types = non-primitive types
- references types typically implemented by classes and objects

- Example 1:   String

- Example 2:   arrays (mutable, fixed-length lists)

Variable Names

- start with a letter (convention: a-z) or underscore "_"
- contain letters a-z and A-Z, digits 0-9, and underscore "_"

- can be any such name except for 50 reserved names:

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Multiple Assignment

- variables can be assigned to different values of the same type:
  - Example: int x = 23;

    x = 42;

  - Instructions are executed top-to bottom => x refers to 42

- variables cannot be assigned to values of different type:
  - Example: int x = 23;

    x = 42.0;   // !ERROR!

- only exception is if types are "compatible":
  - Example: double x = 23.0;

    x = 42;   // :-)

UNIVERSITY OF SOUTHERN DENMARK.DK

Operators & Operands

- Operators represent computations: + * - / ++ --
  - Example: 23+19   day+month*30   2*2*2*2*2*2-22

- Addition "**+**", Multiplication "*****", Subtraction "**-**" as usual
- there is no exponentiation operator to compute x^y
- need to use Math.pow(x, y) write your own function power

static int power(int a, int b) {
    if (b == 0) return 1; else return a*power(a,b-1);
}

- Division "**/**" rounds down integers (differently from Python)
  - Example Java: 3/-2 has value **-1**
  - Example Python: 3/-2 has value **-2**

Boolean Expressions

- expressions of type boolean with value either true or false

- logic operators for computing with Boolean values:
    - x && y   true if, and only if, x is true and y is true
    - x || y   true if (x is true or y is true)
    - !x   true if, and only if, x is false

- Java does NOT treat numbers as Boolean expressions ☺

UNIVERSITY OF SOUTHERN DENMARK.DK

Expressions

- Expressions can be:
    - Values: 42   23.0   "Hej med dig!"
    - Variables: x   y   name1234
    - built from operators: 19+23.0   x*x+y*y
- grammar rule:
    - <expr>   =>   <value> |

        <var> |

        <expr> <operator> <expr> |

        (<expr>)
- every expression has a value:
    - replace variables by their values
    - perform operations

UNIVERSITY OF SOUTHERN DENMARK.DK

Increment and Decrement

- abbreviation for incrementing / decrementing (like in Python)
- Example: counter = counter + 1;

           counter += 1;

- in special case of "+1", we can use "++" operator
- Example: counter++;

- two variants: post- and pre-increment
- Example: int x = 42;

           int y = x++;   // x == 43 && y == 42

           int z = ++y;   // y == 43 && z == 43

- same for decrementing with "--" operator

Relational Operators

- relational operators are operators, whose value is boolean

- important relational operators are:

	Example True	Example False
	Example True	Example False
x < y	23 < 42	'W' < 'H'
x <= y	42 <= 42.0	Math.PI <= 2
x == y	42 == 42.0	2 == 2.00001
x != y	42 != 42.00001	2 != 2.0
x >= y	42 >= 42	'H' >= 'h'
x > y	'W' > 'H'	42 > 42

- remember to use "==" instead of "=" (assignment)!

Conditional Operator

■ select one out of two expressions depending on condition

■ as a grammar rule:

<cond-op>   =>   <cond> ? <expr$_1$> : <expr$_2$>

■ Example:

int answer = (1 > 0) ? 42 : 23;

■ useful as abbreviation for many small if-then-else constructs

UNIVERSITY OF SOUTHERN DENMARK.DK

Operator Precedence

- expressions are evaluated left-to-right
  - Example: 64 - 24 + 2   ==   42

- BUT: like in mathematics, "*" binds more strongly than "+"
  - Example: 2 + 8 * 5   ==   42

- parentheses have highest precedence: 64 - (24 + 2) == 38

  - Parentheses "(<expr>)"
  - Increment "++" and Decrement "--"
  - Multiplication "*" and Division "/"
  - Addition "+" and Subtraction "-"
  - Relational Operators, Boolean Operators, Conditonal, …

String Operations

- Addition "**+**" works on strings; "**-**", "*****", and "**/**" do NOT
- other operations implemented as methods of class String:

```
String s1 = "Hello ";  String s2 = "hello ";
boolean b1 = s1.equals(s2);              // b1 == false
boolean b2 = s1.equalsIgnoreCase(s2);    // b2 == true
int i1 = s1.length();                    // i1 == 5
char c = s1.charAt(1);                   // c == 'e'
String s3 = s1.substring(1,3);           // s3.equals("el")
int i2 = s1.indexOf(s3);                 // i2 == 1
int i3 = s1.compareTo(s2);               // i3 == -1
String s4 = s1.toLowerCase();            // s4.equals(s2)
String s5 = s1.trim();                   // s5.equals("Hello")
```

Formatting Strings

- convert to string using format strings (like in Python)
- Example:

  System.out.println(String.format("%d", 42));

  System.out.printf("%d\n", 42);

- String.format(String s, Object... args) more general

- format sequence %d for integer, %g for float, %s for string

- for multiple values, use multiple arguments
- Example:

  System.out.printf("The %s is %g!", "answer", 42.0);

Statements

- instructions in Java are called *statements*

- so far we know 3 different statements:
  - expression statements: System.out.println("Ciao!");
  - assignments "=":   c = a*a+b*b;
  - return statements: return c;

- as a grammar rule:

  <stmt>   =>   <expr> |

                        <var> = <expr> |

                        return <expr>

Comments

- programs are not only written, they are also read

- document program to provide intuition:
    - Example 1:   c = Math.sqrt(a*a+b*b);   // use Pythagoras
    - Example 2:   int tmp = x; x = y; y = tmp; // swap x and y
- all characters after the comment symbol "//" are ignored

- multiline comments using "/*" and "*/"
- Example: /* This comment
                                      is very long! */

- Javadoc comments using "/**" and "*/"
- Example: /** This function rocks! */

(Syntactic) Differences Java / Python

- every statement is ended by a semi-colon ";"
- Example: import java.util.Scanner;

- indentation is a convention, not a must ☹
- blocks of code are marked by curly braces "{" and "}"
- Example: public class A {public static void main(String[]
args) {Scanner sc = new Scanner(System.in); int a = sc.nextInt();
System.out.println(a*a);}}

- objects are created using "new"
- Java variables require type declarations
- Example: Scanner sc = null; int a = 0; int b; b = 1;