DM537
Object-Oriented Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

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

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;         // :-)
    ```

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 ☺

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

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
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

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;

CALLING & DEFINING FUNCTIONS

Functions and Methods

- all functions in java are defined inside a class
- BUT static functions are not associated with one object

- a static function belongs to the class it is defined in
- functions of a class called by <class>.<function>(<args>)
- Example: `Math.pow(2, 6)`

- all other (i.e. non-static) functions belong to an object
- in other words, all non-static functions are methods!
- functions of an object called by <object>.<function>(<args>)
- Example:

```
String s1 = "Hello!";
System.out.println(s1.toUpperCase());
```

Calling Functions & Returning Values

- function calls are expressions exactly like in Python
- Example:

```
int x = sc.nextInt();
```

- argument passing works exactly like in Python
- Example:

```
System.out.println(Math.log(Math.E))
```

- the return statement works exactly like in Python
- Example:

```
return Math.sqrt(a*a+b*b);
```

Function Definitions

- functions are defined using the following grammar rule:

<func.def> => static <type> <function>(…, <type$_i$> <arg$_i$>, …) {

  <instr$_1$>; …; <instr$_k$>; }

- Example (static function):

```
public class Pythagoras {
    static double pythagoras(double a, double b) {
        return Math.sqrt(a*a+b*b);
    }
    public static void main(String[] args) {
        System.out.println(pythagoras(3, 4));
    }
}
```

Method Definitions

- methods are defined using the following grammar rule:

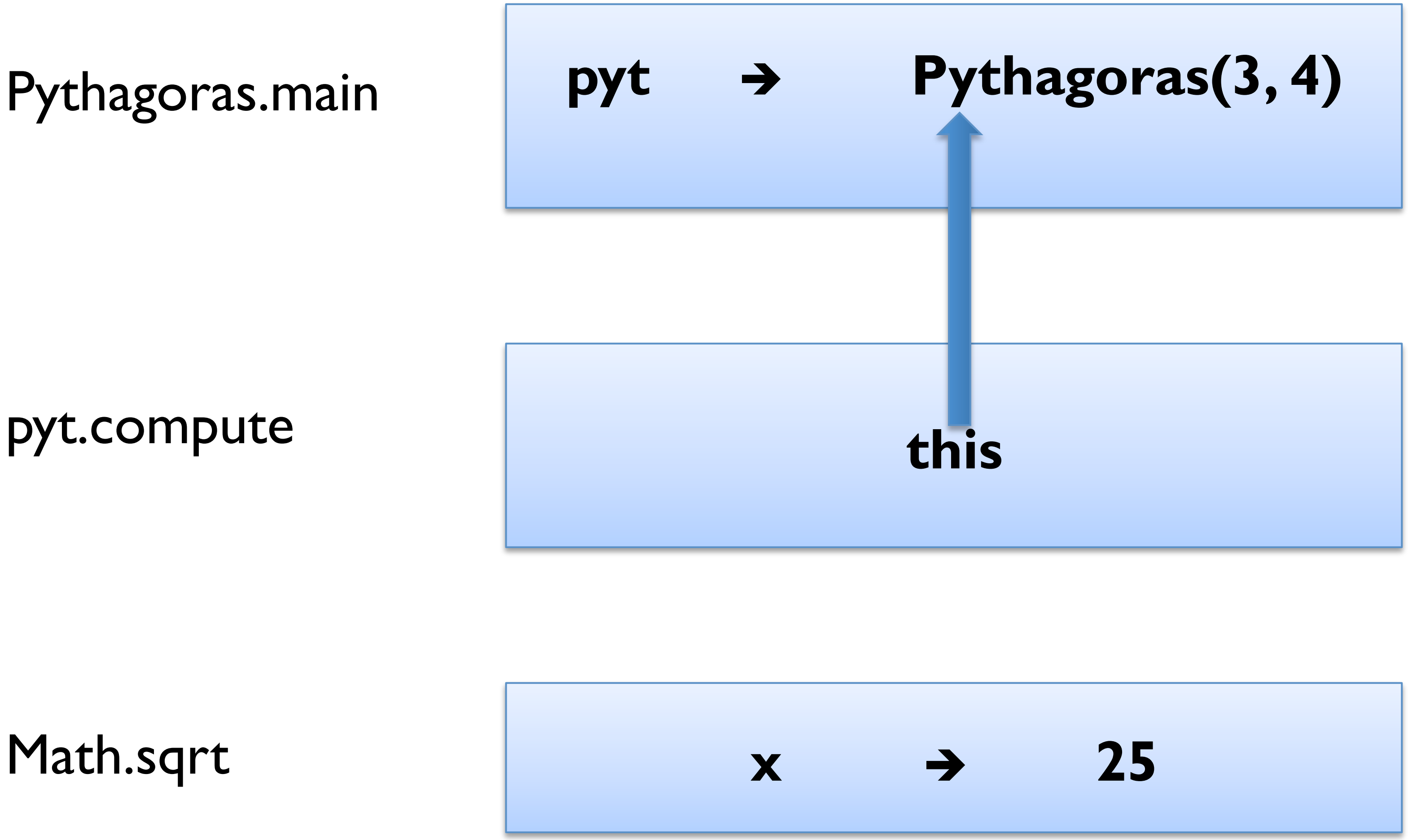
<meth.def> => <type> <function>(…, <type$_i$> <arg$_i$>, …) {
    <instr$_1$>; …; <instr$_k$>; }

- Example (method):

```
public class Pythagoras {
  double a, b;
  Pythagoras(double a, double b) { this.a = a;  this.b = b; }
  double compute() { return Math.sqrt(this.a*this.a+this.b*this.b); }
  public static void main(String[] args) {
    Pythagoras pyt = new Pythagoras(3, 4);
    System.out.println(pyt.compute());
  } }
```

constructor corresponds to __init__(self, a, b)

Stack Diagrams



SIMPLE ITERATION

Iterating with While Loops

- iteration = repetition of code blocks
- while statement:

  <while-loop> => while (<cond>) {

  <instr$_1$>; <instr$_2$>; <instr$_3$>;

  }

- Example:

```
static void countdown(int n) {
  while (n > 0) {
    System.out.println(n);
    n--;
  }
  System.out.println("Ka-Boom!"); }
```

n == 0

true

Breaking a Loop

- sometimes you want to *force* termination
- Example:

```
while (true) {
    System.out.println("enter a number (or 'exit'):\n");
    String num = sc.nextLine();
    if (num.equals("exit")) {
        break;
    }
    int n = Integer.parseInt(num);
    System.out.println("Square of "+n+" is: "+n*n);
}
System.out.println("Thanks a lot!");
```

Approximating Square Roots

- Newton's method for finding root of a function f:
  1. start with some value x_0
  2. refine this value using $x_{n+1} = x_n - f(x_n) / f'(x_n)$
- for square root of a: $f(x) = x^2 - a \quad f'(x) = 2x$
- simplifying for this special case: $x_{n+1} = (x_n + a / x_n) / 2$
- Example:

```
double xn = 1;
while (true) {
    System.out.println(xn);
    double xnp1 = (xn + a / xn) / 2;
    if (xnp1 == xn) {  break;  }
    xn = xnp1;
}
```

Approximating Square Roots

- Newton's method for finding root of a function f:
  1. start with some value x_0
  2. refine this value using $x_{n+1} = x_n - f(x_n) / f'(x_n)$
- for square root of a: $f(x) = x^2 - a$ $f'(x) = 2x$
- simplifying for this special case: $x_{n+1} = (x_n + a / x_n) / 2$
- Example:

```
double xnp1 = 1;
do {
    xn = xnp1;
    System.out.println(xn);
    double xnp1 = (xn + a / xn) / 2;
} while (xnp1 != xn);
```

Iterating with For Loops

- (standard) for loops very different from Python
- grammar rule:

```
<for-loop>    =>  for (<init>; <cond>; <update>) {
                      <instr_1>;  …;  <instr_k>;
                  }
```

- Execution:
  1. initialize counter variable using <init>
  2. check whether condition <cond> holds
  3. if not, END the for loop
  4. if it holds, first execute $<\text{instr}_1>$ … $<\text{instr}_k>$
  5. then execute <update>
  6. jump to Step 2

Iterating with For Loops

- (standard) for loops very different from Python
- grammar rule:

```
<for-loop>        =>  for (<init>; <cond>; <update>) {
                          <instr_1>; ...; <instr_k>;
                      }
```

- Example:

```
int n = 10;
while (n > 0) {
    System.out.println(n);
    n--;
}
System.out.println("Ka-Boom!");
```

Iterating with For Loops

- (standard) for loops very different from Python
- grammar rule:

```
        <for-loop>        =>  for (<init>; <cond>; <update>) {
                                   <instr_1>;  …;  <instr_k>;
                              }
```

- Example:

```
int n = 10;
while (n > 0) {                    for (int n = 10;  n > 0;  n--) {
   System.out.println(n);
   n--;
}                                  }
System.out.println("Boo!");
```

Iterating with For Loops

- (standard) for loops very different from Python
- grammar rule:

```
<for-loop>        =>  for (<init>; <cond>; <update>) {
                          <instr_1>;  …;  <instr_k>;
                      }
```

- Example:

```
int n = 10;
while (n > 0) {                        for (int n = 10;  n > 0;  n--) {
    System.out.println(n);                 System.out.println(n);
    n--;
}                                      }
System.out.println("Boo!");            System.out.println("Boo!");
```

CONDITIONAL EXECUTION

Conditional Execution

- the if-then statement executes code only if a condition holds
- grammar rule:

```
<if-then>        =>     if (<cond>) {
                            <instr_1>;  …;  <instr_k>;
                        }
```

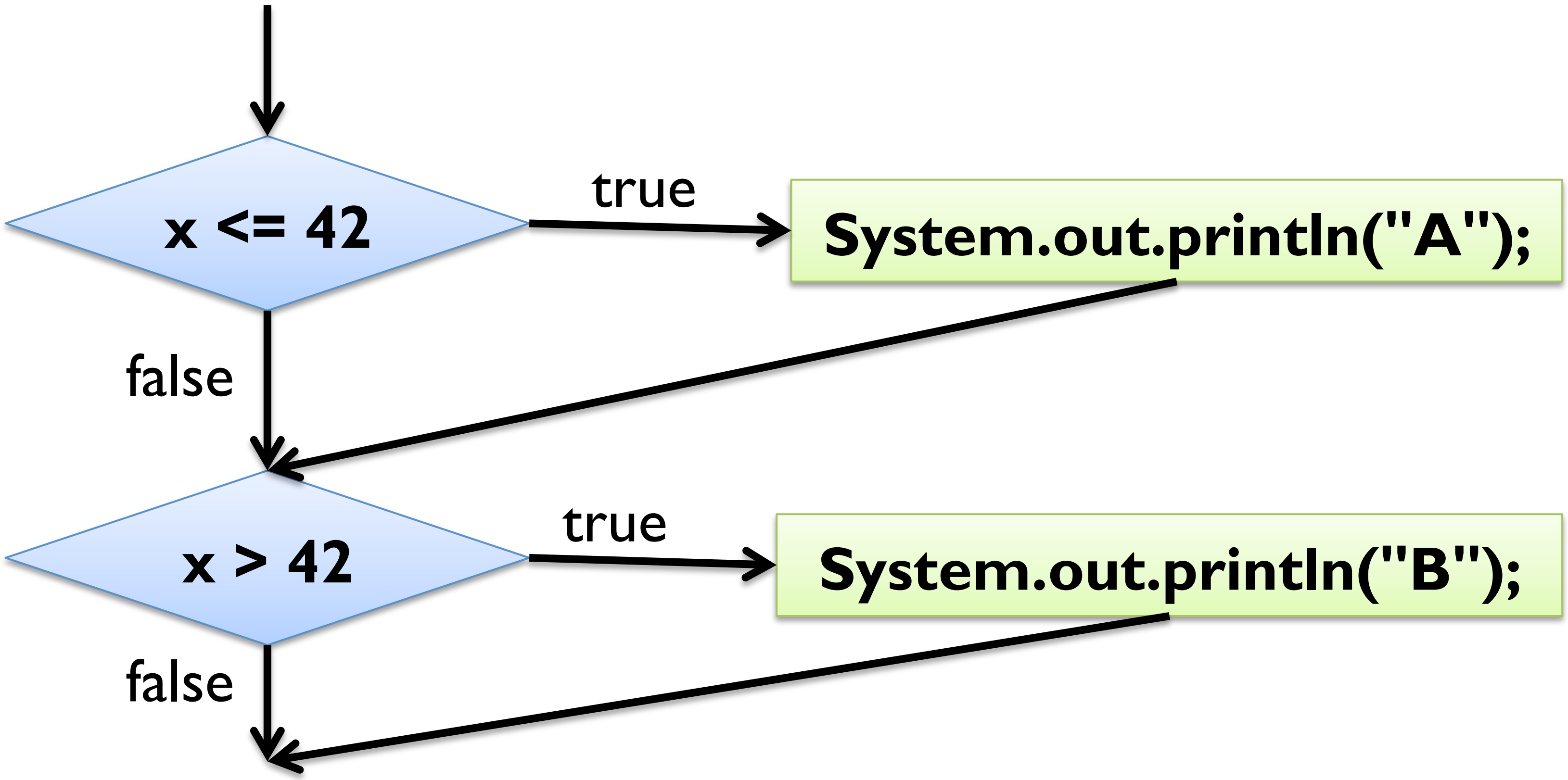
- Example:

```
if (x <= 42) {
   System.out.println("not more than the answer");
}
if (x  >  42) {
   System.out.println("sorry - too much!");
}
```

Control Flow Graph

- Example:

```
if (x <= 42) {  System.out.println("A");  }
if (x  >  42) {  System.out.println("B");  }
```



Alternative Execution

- the if-then-else statement executes one of two code blocks
- grammar rule:

```
<if-then-else>   =>   if (<cond>) {
                          <instr_1>;  …;  <instr_k>;
                      } else {
                          <instr'_1>;  …;  <instr'_k'>;
                      }
```

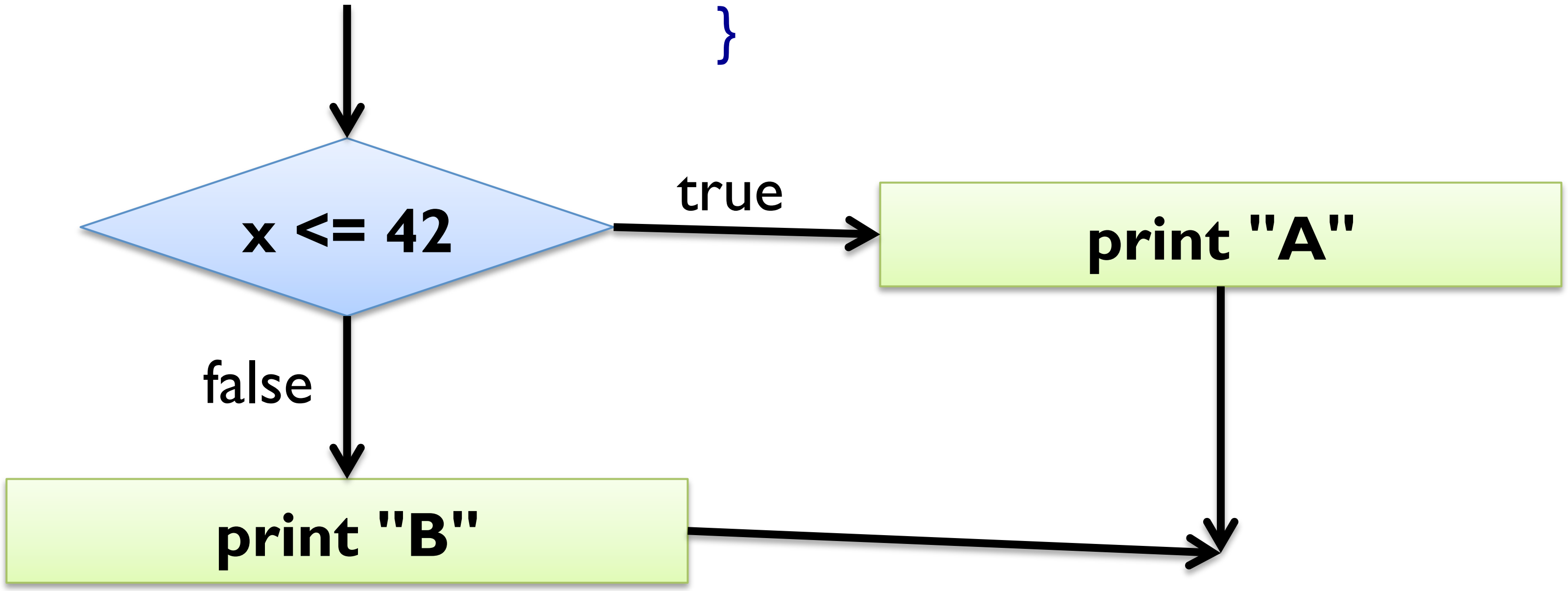
- Example:

```
if (x <= 42) {
   System.out.println("not more than the answer");
} else {
   System.out.println("sorry - too much!");
}
```

Control Flow Graph

- Example:

```
if (x <= 42) {
        System.out.println("A");
} else {
        System.out.println("B");
}
```



Chained Conditionals

- alternative execution a special case of chained conditionals
- grammar rules:

```
<if-chained>    =>    if (<cond_1>) {
                          <instr_1,1>; …; <instr_k1,1>;
                      } else if (<cond_2>) {
                          …
                      } else {
                          <instr_1,m>; …; <instr_km,m>;
                      }
```

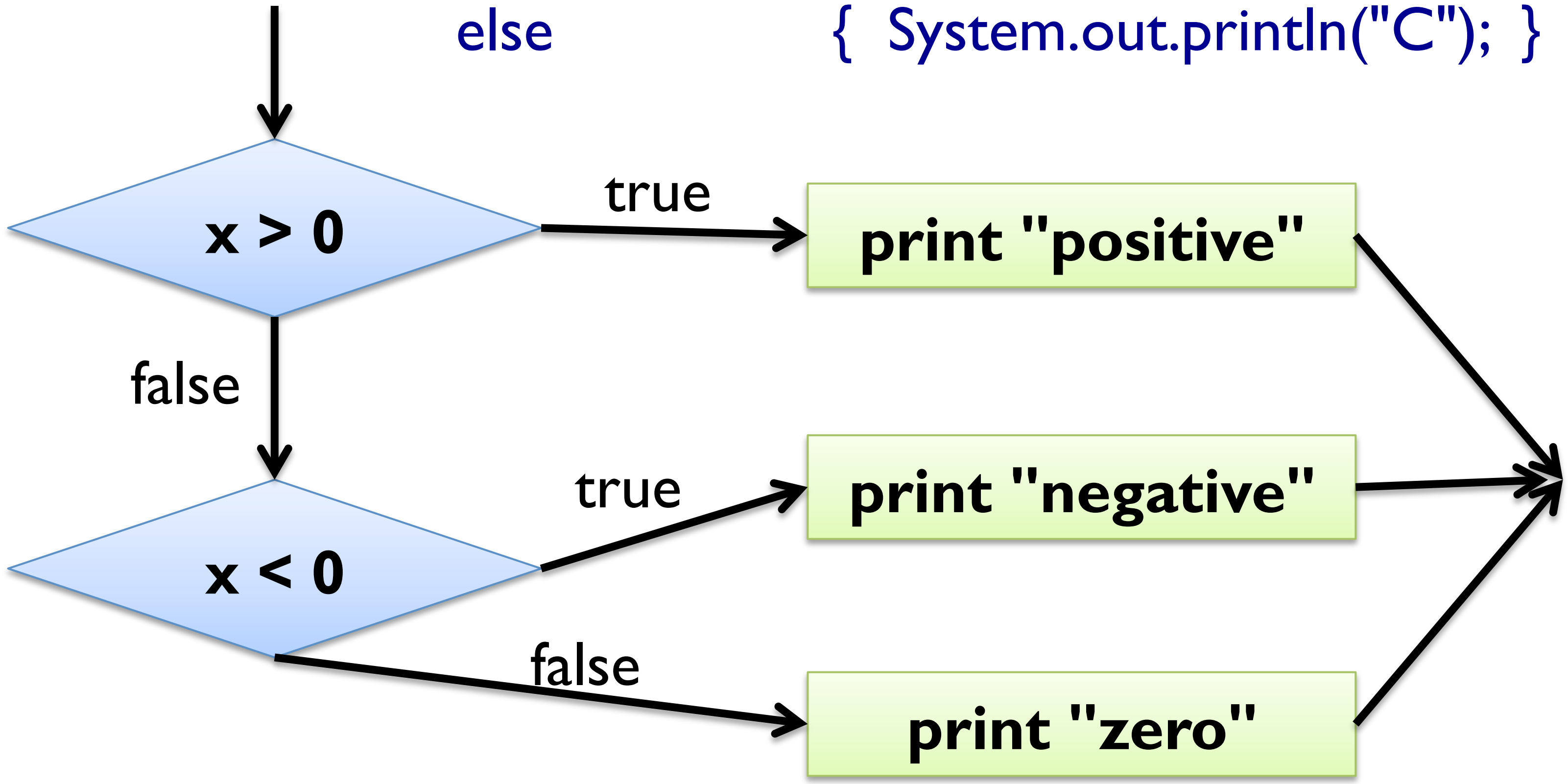
- Example:

```
if (x > 0)        {  System.out.println("positive");  }
else if (x < 0)   {  System.out.println("negative");  }
else              {  System.out.println("zero");  }
```

Control Flow Diagram

- Example:

```
if (x > 0)        { System.out.println("A"); }
else if (x < 0)   { System.out.println("B"); }
else              { System.out.println("C"); }
```



Switch Statement

- for int and char, special statement for multiple conditions
- grammar rules:

```
<switch>    =>    switch (<expr>) {
                  case <const_1>:
                      <instr_1,1>; …; <instr_k1,1>;
                      break;
                  case <const_2>:
                       …
                  default:
                       <instr_1,m>; …; <instr_km,m>;
                  }
```

Switch Statement

- Example:

```
int n = sc.nextInt();
switch (n) {
case 0:
    System.out.println("zero");
    break;
case 1:
case 2:
    System.out.println("smaller than three");
default:
    System.out.println("negative or larger than two");
}
```

Nested Conditionals

- conditionals can be nested below conditionals:

```
if (x > 0) {
        if (y > 0)          { System.out.println("Quadrant 1"); }
        else if (y < 0)     { System.out.println("Quadrant 4"); }
        else                { System.out.println("positive x-Axis"); }
} else if (x < 0) {
        if (y > 0)          { System.out.println("Quadrant 2"); }
        else if (y < 0)     { System.out.println("Quadrant 3"); }
        else                { System.out.println("negative x-Axis"); }
} else  { System.out.println("y-Axis"); }
```

TYPE CASTS & EXCEPTION HANDLING

Type Conversion

- Java uses *type casts* for converting values
- (int) x: converts x into an integer
  - Example 1: ((int) 127) + 1 == 128
  - Example 2: (int) -3.999 == -3
- (double) x: converts x into a float
  - Example 1: (double) 42 == 42.0
  - Example 2: (double) "42" results in Compilation Error
- (String) x: views x as a string
  - Example: Object o = "Hello World!”;
    String s = (String) o;

Catching Exceptions

- type conversion operations are error-prone
- Example:
```
Object o = new Integer(23);
Strings s = (String) o;
```

- good idea to avoid type casts
- sometimes necessary, e.g. when implementing equals method

- use try-catch statement to handle error situations
- Example 1:
```
String s;
try {
    s = (String) o;
} catch (ClassCastException e) {
    s = "ERROR";  }
```

Catching Exceptions

- use try-catch statement to handle error situations
- Example 2:

```
try {
    double x;
    x = Double.parseDouble(str);
    System.out.println("The number is " + x);
} catch (NumberFormatException e) {
    System.out.println("The number sucks.");
}
```

Arrays

- array = built-in, mutable list of fixed-length
- type declared by adding "[]" to base type
- Example: `int[] speedDial;`

- creation using same "new" as for objects
- size declared when creating array
- Example: `speedDial = new int[20];`

- also possible to fill array using "{}" while creating it
- then length determined by number of filled elements
- Example: `speedDial = {65502327, 55555555};`

Arrays

- array = built-in, mutable list of fixed-length
- access using "[index]" notation (both read and write, 0-based)
- size available as attribute ".length"
- Example:

```
int[] speedDial = {65502327, 55555555};
for (int i = 0; i < speedDial.length; i++) {
    System.out.println(speedDial[i]);
    speedDial[i] += 100000000;
}
for (int i = 0; i < speedDial.length; i++) {
    System.out.println(speedDial[i]);
}
```

Command Line Arguments

- command line arguments given as array of strings
- Example:

```
public class PrintCommandLine {
    public static void main(String[] args) {
        int len = args.length;
        System.out.println("got "+len+" arguments");
        for (int i = 0; i < len; i++) {
            System.out.println("args["+i+"] = "+args[i]);
        }
    }
}
```

Reading from Files

- done the same way as reading from the user
- i.e., using the class java.util.Scanner
- instead of System.in we use an object of type java.io.File
- Example (reading a file given as first argument):

```
import java.util.Scanner;  import java.io.File;
public class OpenFile {
    public static void main(String[] args) {
        File infile = new File(args[0]);
        Scanner sc = new Scanner(infile);
        while (sc.hasNext()) {
            System.out.println(sc.nextLine());
}   }   }
```

Reading from Files

- Example (reading a file given as first argument):

```
import java.util.Scanner;  import java.io.*;
public class OpenFile {
    public static void main(String[] args) {
        File infile = new File(args[0]);
        try {
            Scanner sc = new Scanner(infile);
            while (sc.hasNext()) { System.out.println(sc.nextLine()); }
        } catch (FileNotFoundException e) {
            System.out.println("Did not find your strange "+args[0]);
}   }   }
```

Writing to Files

- done the same way as writing to the screen
- i.e., using the class java.io.PrintStream
- System.out is a predefined java.io.PrintStream object
- Example (copying a file line by line):

```
import java.io.*;  import java.util.Scanner;
public class CopyFile {
   public static void main(String[] args) throws new
FileNotFoundException {
      Scanner sc = new Scanner(new File(args[0]));
      PrintStream target = new PrintStream(new File(args[1]));
      while (sc.hasNext()) {  target.println(sc.nextLine());  }
      target.close();  }   }
```

Throwing Exceptions

- Java uses throw (comparable to raise in Python)
- Example (method that receives unacceptabe input):

```
static double power(double a, int b) {
    if (b < 0) {
        String msg = "natural number expected";
        throw new IllegalArgumentException(msg);
    }
    result = 1;
    for (; b > 0; b--) {  result *= a;  }
    return result;
}
```