DM550 / DM857
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM550/

http://imada.sdu.dk/~petersk/DM857/

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$_i$>, …) {

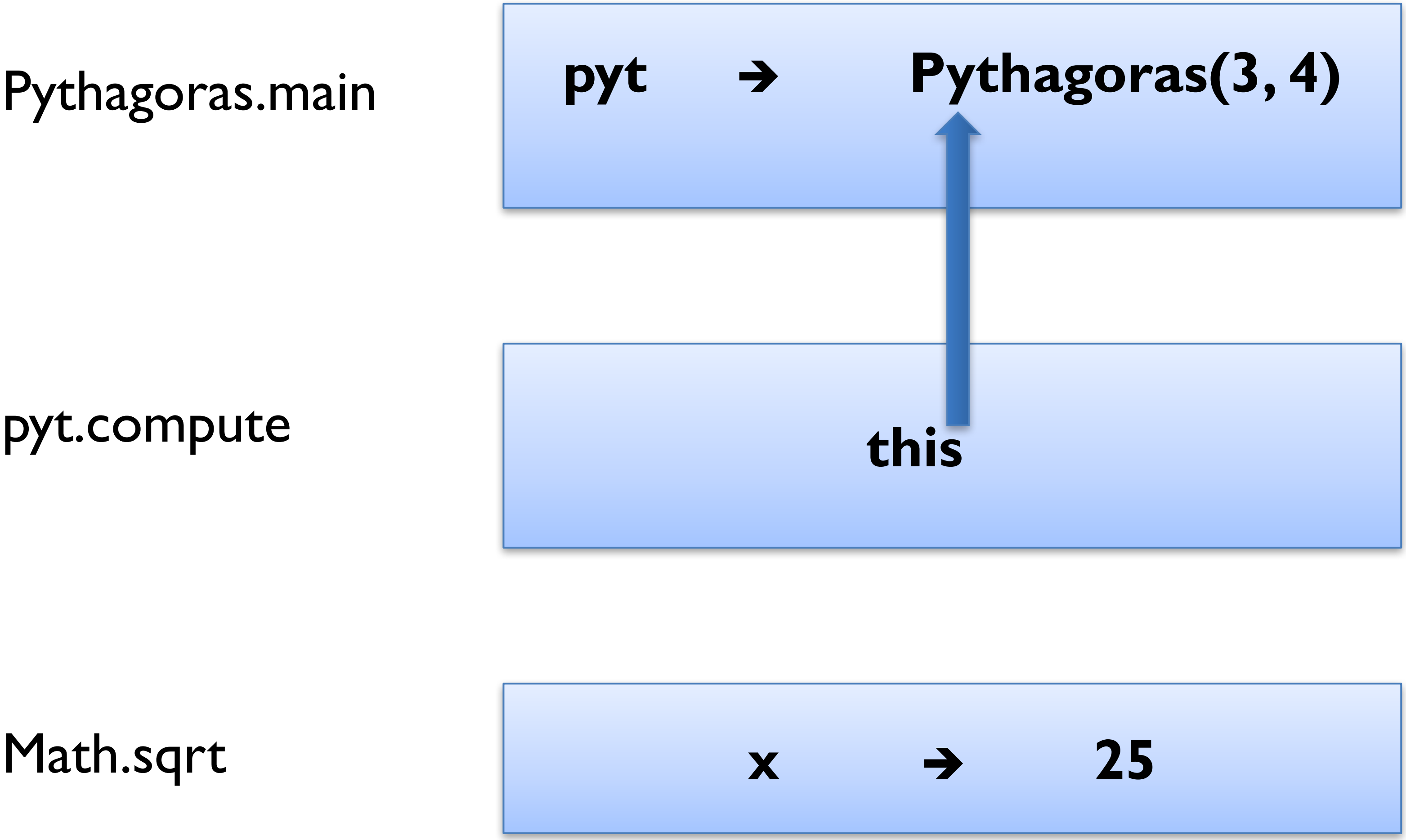
<instr$_1$>; …; <instr$_k$>; }

- 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



Pythagoras.main

pyt → Pythagoras(3, 4)

pyt.compute

this

Math.sqrt

x → 25

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!");  }
```

true

n == 0

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: $f(x) = x^2 – a$ $f’(x) = 2x$
- 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 <instr$_1$> … <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) {
    System.out.println(n);
    n--;
}
System.out.println("Boo!");
```

```
for (int n = 10;  n > 0;  n--) {
    System.out.println(n);

}
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$_k$>;
                       }
```

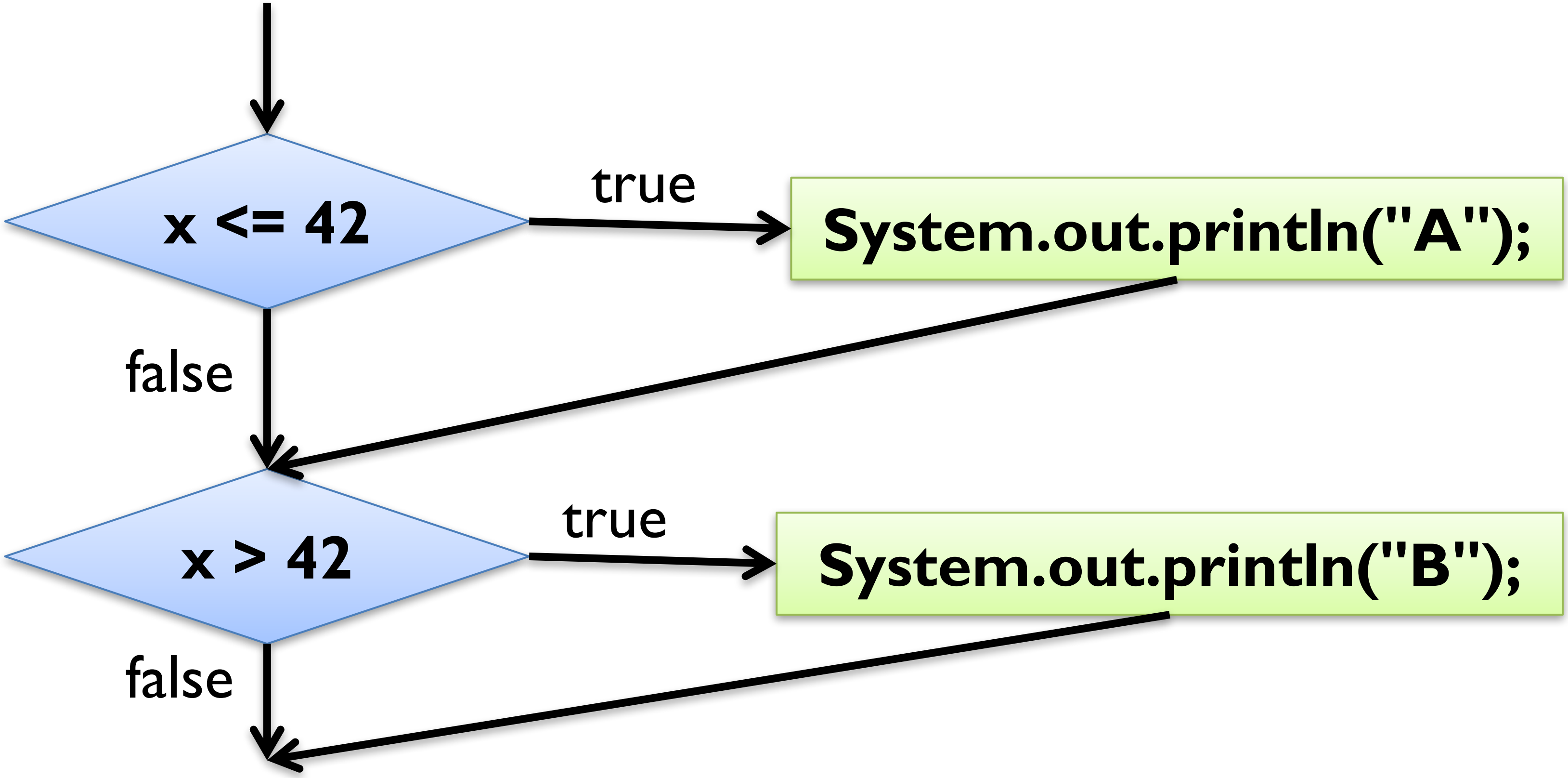
- Example:

```
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>) {
$\langle \text{instr}_1 \rangle$; …; $\langle \text{instr}_k \rangle$;
} else {
$\langle \text{instr'}_1 \rangle$; …; $\langle \text{instr'}_{k'} \rangle$;
}

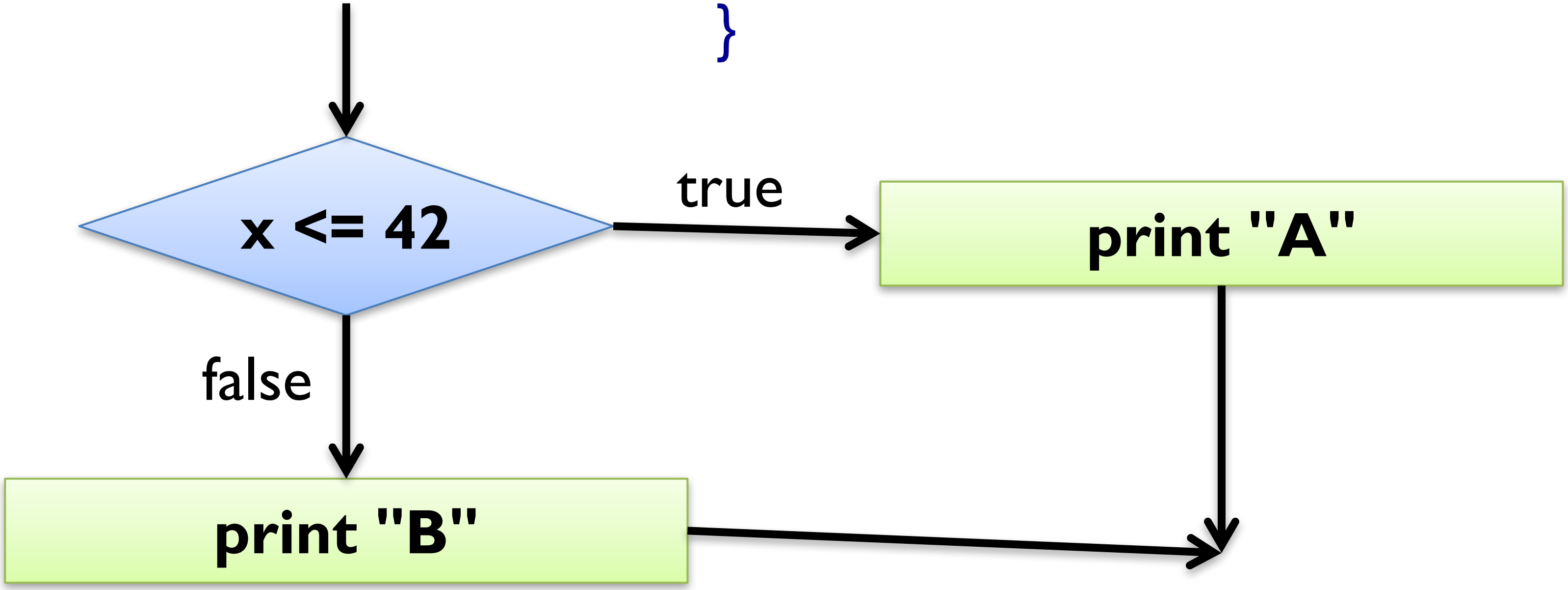
- Example:

```
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>) {
  <$\text{instr}_{1,1}$>; …; <$\text{instr}_{k1,1}$>;
  } else if (<cond_2>) {
  …
  } else {
  <$\text{instr}_{1,m}$>; …; <$\text{instr}_{km,m}$>;
  }

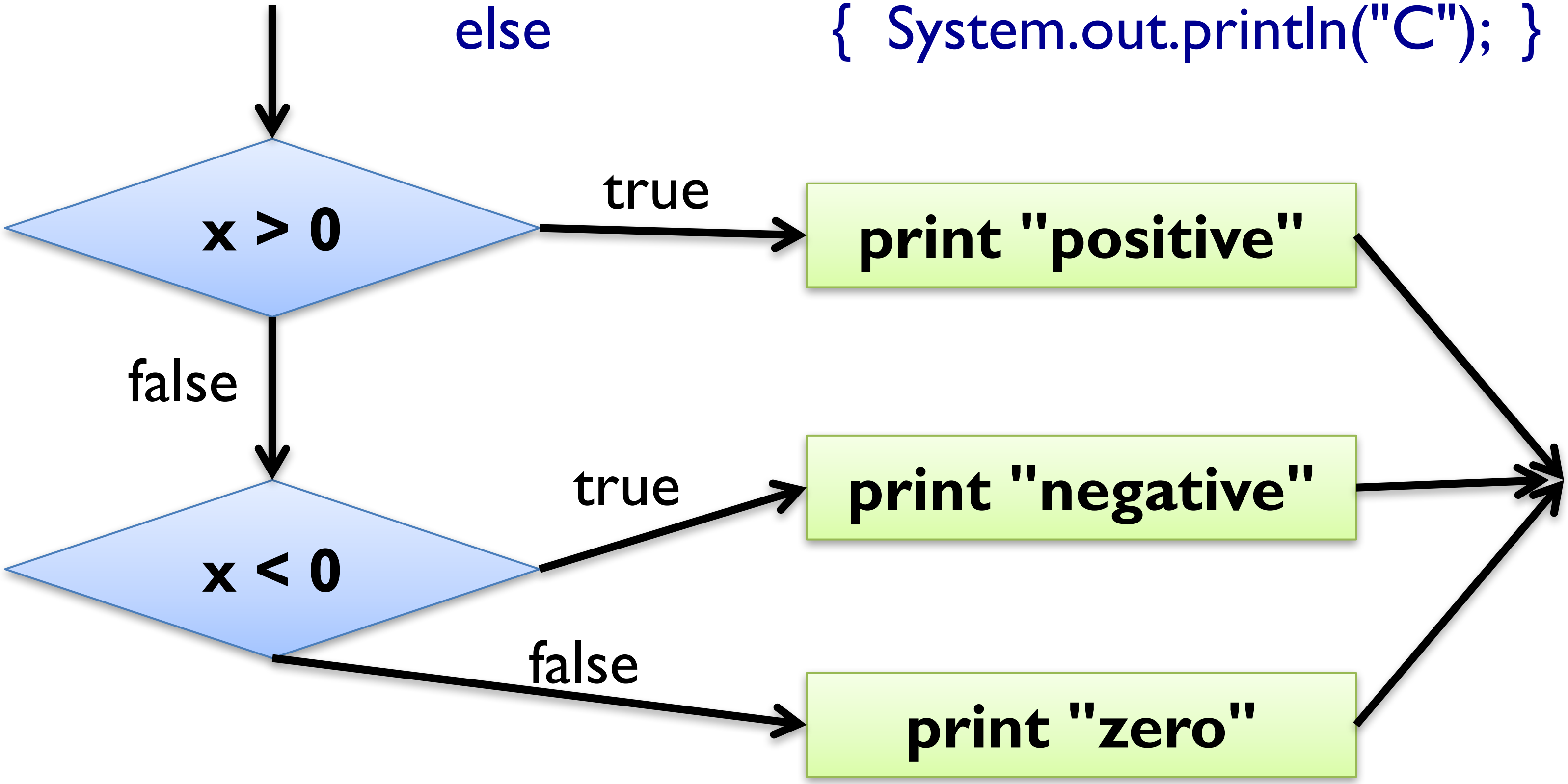
- Example:

```
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>:
<$\text{instr}_{1,1}$>; …; <$\text{instr}_{k1,1}$>;
break;
case <const_2>:
…
default:
<$\text{instr}_{1,m}$>; …; <$\text{instr}_{km,m}$>;
}

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};