Five-Minute Review

1. What are *expression statements*? *Compound statements*?
2. What is a *scope*?
3. What are *conditional statements* in Java? How about *iterative statements*?
4. In conditionals, why should we use compound statements instead of simple statements?
5. What is the *repeat-until-sentinel* pattern?

Guide to Success I

Throughout the semester:

1. Between lectures, go through **slides**, including those not shown in class
2. For the material covered in the **book** (about 80% of class content), work through corresponding book chapter, either before or after material is covered in class
3. Align **programming** tasks with concepts in class/book; ask for help whenever needed, but write your own code
4. At the end of each lecture/chapter, *first* answer **review questions** in book yourself, *then* compare with solutions
5. Participate in **practice exam** (last *Globalübung* in December)
6. Finally: **JUST DO IT.**

JUST DO IT.

- Becoming a programmer is like learning to ride a bike – it does not suffice to watch other bikers, you just have to get on the bike yourself and start pedaling.
- Thus, to become a programmer, *and* to pass the exam well, "studying" ("*Lernen*") alone will most likely not do the trick – instead, you also need to spend plenty of time with **actively** programming **yourself**.
- At the same time, to become a *good* programmer, you should study other people's code as well and try to get advice from experts whenever possible.
- Same with Math, by the way – if you just stare at the lecture material and problem solutions, you probably won't progress well. Instead, you have to wreck your own brain with trying to solve practice problems, *and* take expert advice.

Guide to Success II

Get connected, join social media

Form study groups (ideal size: 2 – 3)

1. Ask each other if anything is unclear
2. Ask each other 5-Minute Review Questions from slides, in random order
3. Ask each other review questions from book, in random order; compare with solutions (slide notes)
4. Ask each other questions on program assignments

Guide to Success III

When studying for practice/final exam:

1. **Start on time**
2. Re-read chapter summaries
3. Go through lecture slides
4. Write notes, condense them to one page

Programming – Lecture 5

Methods (Chapter 5)

- Message paradigm
- Functions, **`Math`** class
- Writing methods
- Mechanics of method calls
- Decomposition, train example

Recall (Chapter 2): Sending Messages to Objects

receiver **.** *name* **(** *arguments* **) ;**

```
public class HelloProgram extends GraphicsProgram {
   public void run() {
      GLabel label = new GLabel("hello, world", 100, 75);
      label.setFont("SansSerif-36");
      label.setColor(Color.RED);
      add(label);
   }
}
```

label

hello, world

skip simulation

Methods

Method call:

receiver . *name* (*arguments*) ;

Method definition:

modifier *type* *name* (*parameter list*) {

statements in the method body

}

- *Calling*, *returning*, *result*
- *Arguments* / *actual parameters*:
  expressions passed in method call
- *Parameters* / *formal parameters*:
  variables declared in method declaration
- Method *signature*: name + parameters (but not return type)

Methods

- Information hiding
- Methods vs. programs
- Role in expressions
- (*Instance*) *methods* – associated with objects (the default)
- *Static methods* – associated with class, denoted **`static`**

`Math` Class

Math.abs(x**)**	Returns the absolute value of x
Math.min(x**,** y**)**	Returns the smaller of x and y
Math.max(x**,** y**)**	Returns the larger of x and y
Math.sqrt(x**)**	Returns the square root of x
Math.log(x**)**	Returns the natural logarithm of x ($\log_e x$)
Math.exp(x**)**	Returns the value of e raised to the x power (e^x)
Math.pow(x**,** y**)**	Returns the value of x raised to the y power (x^y)
Math.sin(*theta***)**	Returns the sine of *theta*, measured in radians
Math.cos(*theta***)**	Returns the cosine of *theta*
Math.tan(*theta***)**	Returns the tangent of *theta*
Math.asin(x**)**	Returns the angle whose sine is x
Math.acos(x**)**	Returns the angle whose cosine is x
Math.atan(x**)**	Returns the angle whose tangent is x
Math.toRadians(*degrees***)**	Converts an angle from degrees to radians
Math.toDegrees(*radians***)**	Converts an angle from radians to degrees

Aside: Efficiency

```
for (int i = 0;
      i <= Math.pow(2, n + 1) - 1;
      i++) ...
```

Problems:

- Re-computes upper loop bound on every iteration
- Unneeded, costly method call to math library

```
int i_cnt = 1 << (n + 1);
for (int i = 0; i < i_cnt; i++) ...
```

Coding Advice:

- Pre-compute upper loop bound
- Use shift operation for fast integer op. when possible

Coding Advice – Use Power of Shift

For integers **`i`** and **`n`**:

`1 << n`

corresponds to 2^n

and is typically much faster than **`Math.pow(2, n)`**

E.g. **`1 << 10`** corresponds to **`1024`**

`i << n`

corresponds to $i * 2^n$

E.g. **`x << 3`** corresponds to **`x * 8`**

`i >> n`

corresponds to $i / 2^n$

As always, must keep sign/overflow issues in mind

`return` [*expression*]**`;`**

```
private int max(int x, int y) {
    if (x > y) {
        return x;
    } else {
        return y;
    }
}
```

procedures: methods with type **`void`**

- no expression in return statement
- implicit **`return`** at end of method

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Nonnumeric Methods

```
private String weekdayName(int day) {
    switch (day) {
        case 0: return "Sunday";
        case 1: return "Monday";
        case 2: return "Tuesday";
        case 3: return "Wednesday";
        case 4: return "Thursday";
        case 5: return "Friday";
        case 6: return "Saturday";
        default: return "Illegal weekday";
    }
}
```

Note: no `break` required after return

Methods Returning Graphical Objects

```
private GOval createFilledCircle
   (int x, int y, int r, Color color) {
    GOval circle = new GOval(x - r,
        y - r, 2 * r, 2 * r);
    circle.setFilled(true);
    circle.setColor(color);
    return circle;
 }
```

Predicate Methods

```
private boolean isDivisibleBy(
        int x, int y) {
    return x % y == 0;
}
```

```
for (int i = 1; i <= 100; i++) {
    if (isDivisibleBy(i, 7)) {
        println(i);
    }
}
```

```
for (int i = 1; i <= 100; i++) {
    if (isDivisibleBy(i, 7)
          == true) {
        println(i);
    }
}
```

Write instead:

```
for (int i = 1; i <= 100; i++) {
    if (isDivisibleBy(i, 7)) {
        println(i);
    }
}
```

Coding Advice – Booleans

Avoid comparisons with boolean literals!

Bad: `if (flag == true)`

Bad: `if (flag != false)`

Good: `if (flag)`

Bad: `if (flag == false)`

Bad: `if (flag != true)`

Good: `if (!flag)`

```
private boolean isDivisibleBy(
        int x, int y) {
    if (x % y == 0) {
        return true;
    } else {
        return false;
    }
}
```

Coding advice: don't re-compute booleans!

```
private boolean isDivisibleBy(
        int x, int y) {
    return x % y == 0;
}
```

Testing Powers of Two

```
private boolean isPowerOfTwo(int n) {
    if (n < 1) {
        return false;
    }
    while (n > 1) {
        if (n & 1 == 1) {
            return false;
        }
        n >>= 1;
    }
    return true;
}
```

Mechanics of Method Calling

1. Evaluate arguments
2. Copy arg values to parameters,
   in *stack frame*
3. Execute statements in method body
4. **`return`** substitutes value in place of call
5. Discard stack frame, return from callee to caller

How many ways are there to select two coins?

penny + nickel
penny + dime
penny + quarter
penny + dollar

nickel + dime
nickel + quarter
nickel + dollar

dime + quarter
dime + dollar

quarter + dollar

$$C(n, k) = \frac{n\,!}{k\,! \times (n - k)\,!}$$

```
private int combinations(
        int n, int k) {
    return factorial(n) /
        (factorial(k) * factorial(n - k));
}
```

```java
public void run() {
   int n = readInt("Enter number of objects in the set (n): ");
   int k = readInt("Enter number to be chosen (k): ");
   println("C(" + n + ", " + k + ") = " + combinations(n, k) );
}
```

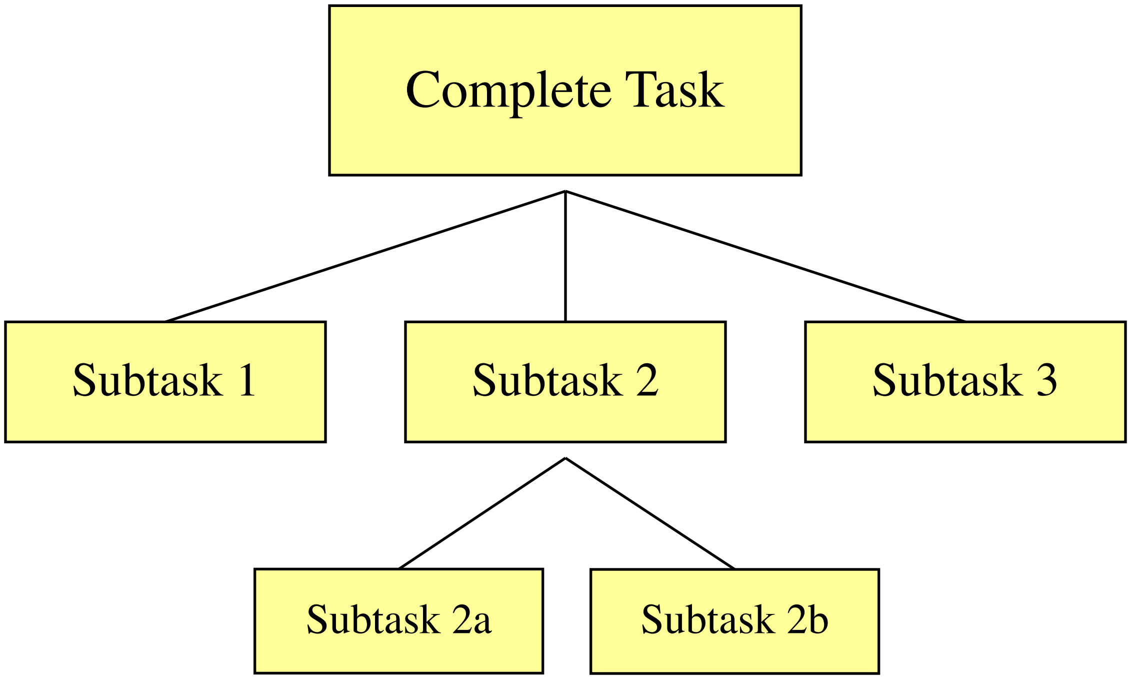
n	k
5	2

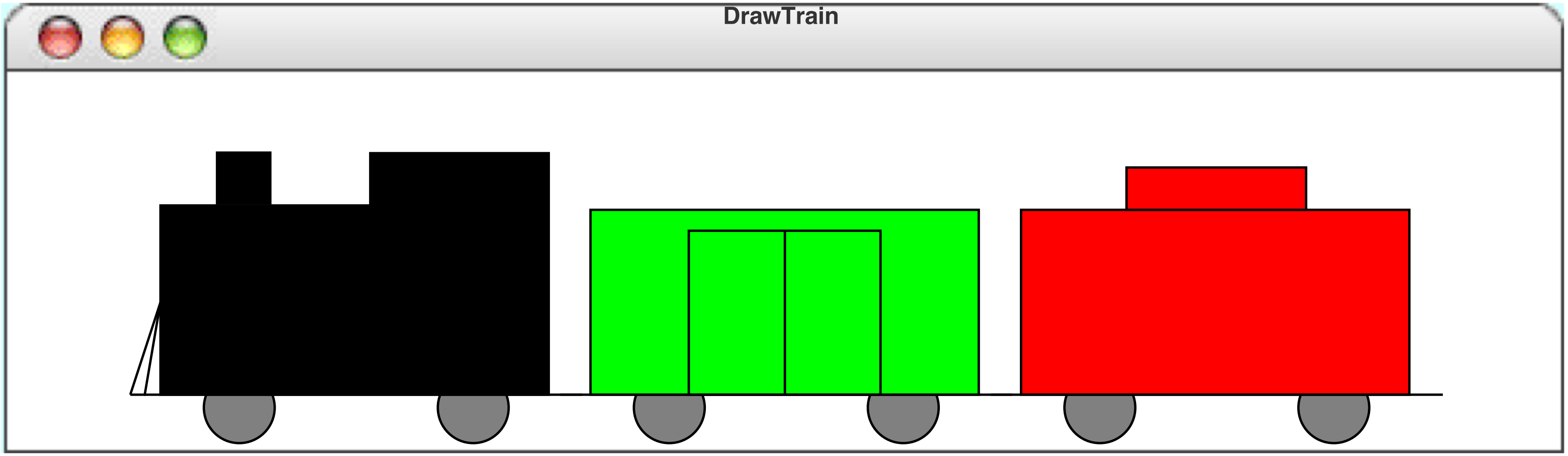
Combinations

```
Enter number of objects in the set (n): 5
Enter number to be chosen (k): 2
C(5, 2) = 10
```


skip simulation

Decomposition



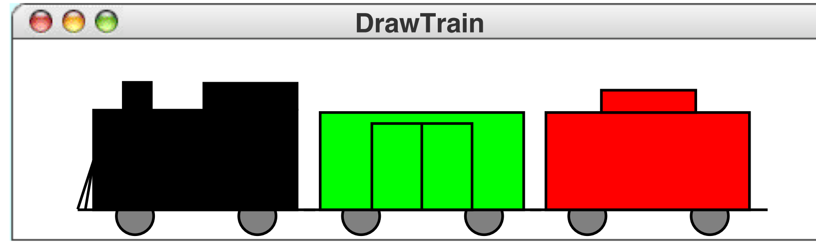



```
public void run() {
```
Draw the engine.

Draw the boxcar.

Draw the caboose.
```
}
```

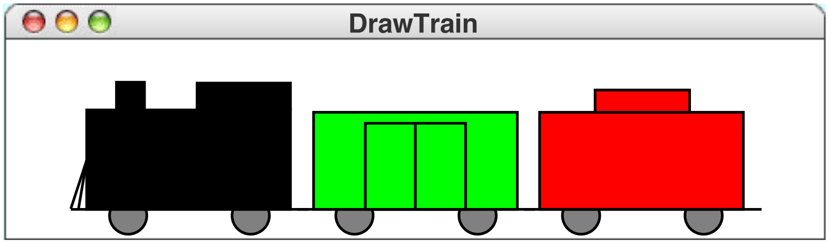
Parameters



Assumptions

- Caller supplies location of each car
- Train cars are same size, have same structure
- Engines are black
- Boxcars come in many colors
- Cabooses are red

```
private void drawEngine(
    double x, double y)
private void drawBoxcar(
    double x, double y, Color color)
private void drawCaboose(
    double x, double y)
```

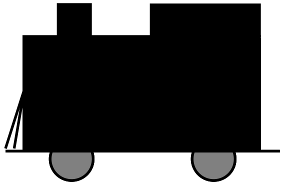


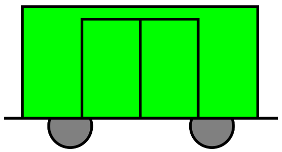
Commonalities

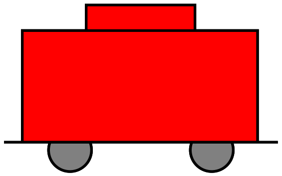
- Frame, wheels, connector
- Drawn by **`drawCarFrame`**

Differences

- Engine: black, adds smokestack, cab, cowcatcher
- Boxcar: colored as specified by caller, adds doors
- Caboose: red, adds cupola.

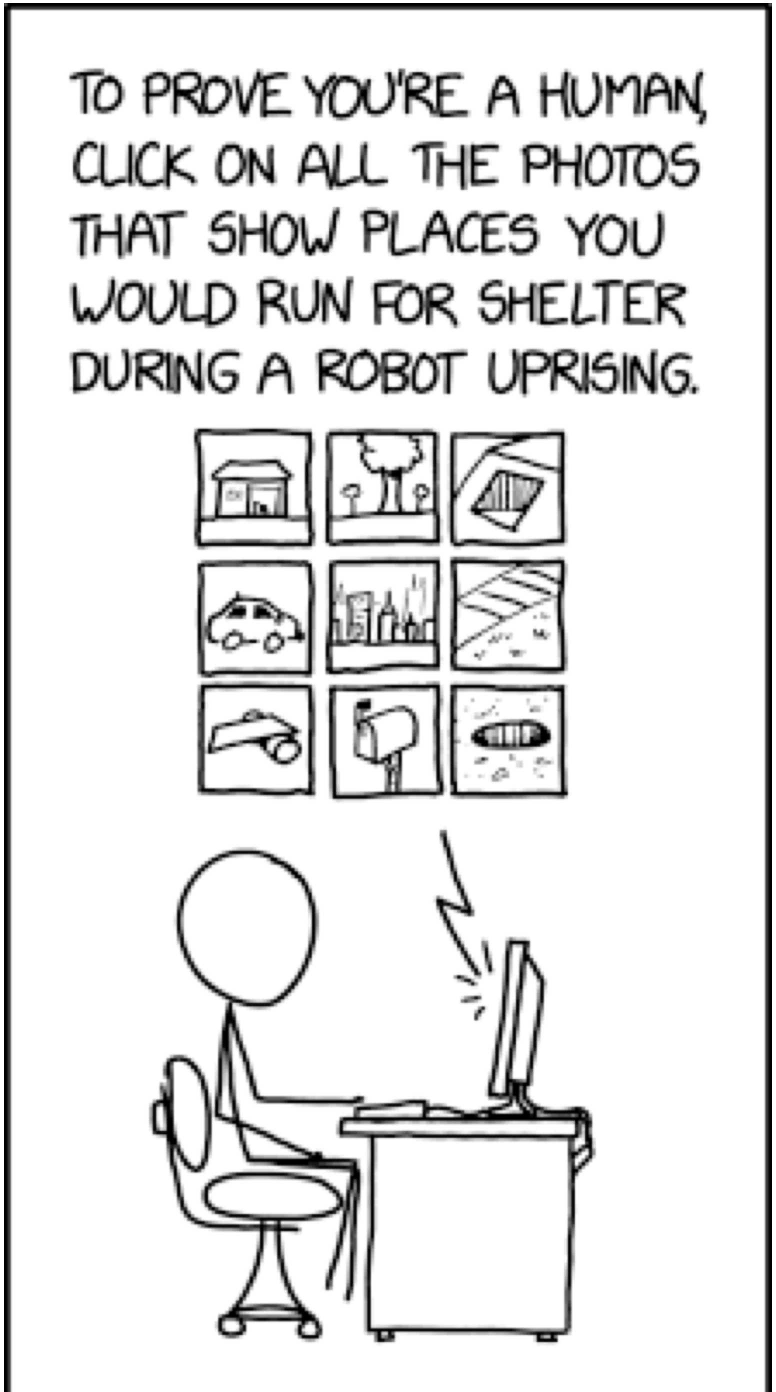






Summary

- Motivation for methods:
  - Code re-use
  - Decomposition
  - Information hiding
- Mental models of method call:
  - Message exchange
  - Functional evaluation
- Different methods may have local variables with same name
- Method calls usually involve a receiving object; `static` methods are associated only with class



Please visit pingo.upb.de/ 643250

https://xkcd.com/2228