

Programs, Statements, Variables

Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)

Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.jku.at

<http://www.risc.jku.at>

General

- Documentation:
 - Java language specification: all details of the programming language.
 - Java Platform API specification: all standard class libraries for input/output, graphics, ... (API: application programming interface).
- Various programming environments:
 - Free: Java Development Kit (JDK), Eclipse, Netbeans, ...
 - Commercial: JDeveloper, IntelliJ, ...
- Two kinds of Java programs:
 - **Applications**: standalone programs.
 - **Applets**: programs embedded into Web pages (outdated technology).

We focus on Java applications in this course.

Java Source Code

```
public class Name
{
    public static void main(String[] args)
    {
        ...
    }
}
```

- Class *Name* .

- *Name* is a Java identifier, e.g. Prog_2A.

- Method *main*:

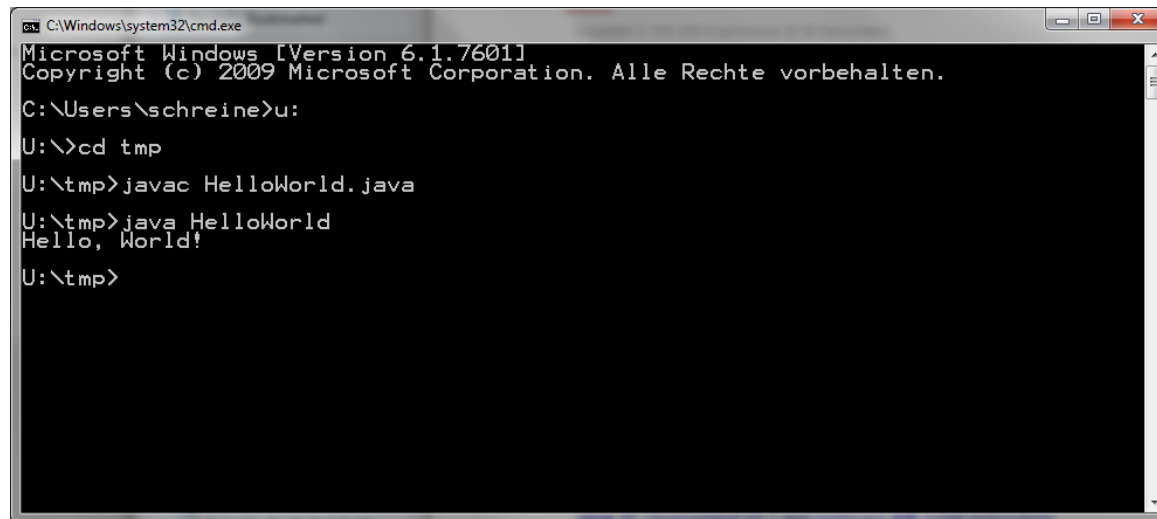
- Body { ... } contains statements executed by program.

- File *Name*.java

```
javac Name.java
java Name
```

Example

```
public class HelloWorld // HelloWorld.java
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.
C:\Users\schreine>u:
U:\>cd tmp
U:\tmp>javac HelloWorld.java
U:\tmp>java HelloWorld
Hello, World!
U:\tmp>
```

Source Code Format

Format does not have any logical significance

```
public class Name {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

```
public class Name { public static void main ( String [ ] args ) { ... } }
```

- May insert/remove empty space or lines.
- Only sequence of **tokens** matters:
 - public, class, *Name*, {, ...

Source code must be readable by other people.

Source Code Conventions

- Use **indentation** to exhibit program structure:
 - After token {, start a new line and indent it by e.g. 2 characters.
- Do not use more than 78 characters per line.
 - Break a longer line into two lines.
- Use **English** for identifiers.
 - Foreign programmers may have to read your source code.
- Write **comments**.
 - Descriptions in natural language (English).
 - Ignored by compiler, help humans to understand program.
 - Most important: class and method headers.

Find your own style and stick to it consistently.

Comments

```
// -----
// Example.java
// Show the general structure of a Java application
//
// Author:  Wolfgang Schreiner <Wolfgang.Schreiner@fhs-hagenberg.ac.at>
// Created: August 18, 2001
// Changed: August 19, 2001
//   Changed the name of the class.
// -----
public class Example
{
    // -----
    // print a sample string to the standard output stream
    // -----
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Basic Text Output

- Method body consists of sequence of **statements**.
 - Each statement terminated by a semicolon (;).
 - Compiled to one or more machine instructions.
 - Executed in the order in which they occur.
 - Effect which is internal to program or influences external environment.

- Output statements:

```
System.out.print(string)
```

```
System.out.println(string)
```

- `print`: prints a string (sequence of characters) to the **standard output** (typically: terminal).
- `println`: prints the string and starts a new line afterwards.

Can write programs that generate multiple lines of output.

Example

- Main Method:

```
public static void main(String[] args)
{
    System.out.print("One, ");
    System.out.print("two, ");
    System.out.println("three.");
    System.out.println("We are done.");
}
```

- Output:

```
One, two, three.
We are done.
```

Printing of Integers

- May print integer numbers and strings:

```
System.out.print("1 plus 1 is ");  
System.out.print(2);  
System.out.println(".");
```

- Output:

```
1 plus 1 is 2.
```

- Simpler form:

```
System.out.println(1 + " plus " + 1 + " is " + 2 + ".");
```

- Compiler converts number 2 to string "2".
- Operator + concatenates strings.
- Same output as above.

String Literals

- Character " delimits string literals.
- Use **escape sequence** \" to include " in literal:

– Statement:

```
System.out.println("He said \"Hello\" and left.");
```

– Output:

```
He said "Hello" and left.
```

- Also **escape character** \ has to be escaped:

– Statement:

```
System.out.println("C:\\windows\\system");
```

– Output:

```
C:\windows\system
```

More Escape Sequences

- New line character `\n` and tabulator `\t`:

- Statement:

- ```
System.out.println("He said:\n\tYes.\nShe said:\n\tNo.");
```

- Output:

- ```
He said:
```

- ```
 Yes.
```

- ```
She said:
```

- ```
 No.
```

Various other escape sequences (see reference manual).

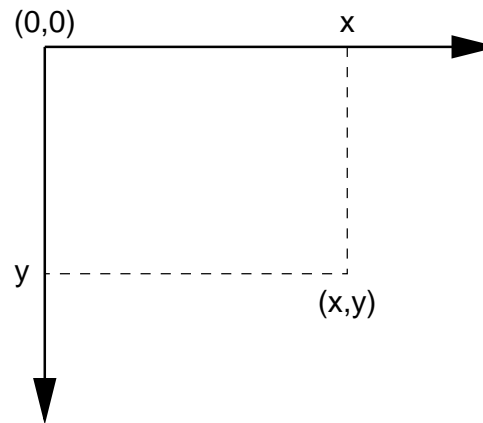
## Basic Graphical Output

- Simple framework provided by file `kwm.jar`:

```
import kwm.*;
import java.awt.*;
public class Name {
 public static void main(String[] args) {
 Drawing.begin("title", width, height);
 Drawing.graphics.operation(...);
 ...
 Drawing.end();
 }
}
```

- `Drawing.graphics.operation(...)`;
  - Statements to paint graphics on the screen.

## Graphical Coordinate System



- Coordinate units are given in screen pixels.

Second coordinate grows from top to bottom.

## Graphical Operations

| Operation                             | Description                                     |
|---------------------------------------|-------------------------------------------------|
| <code>drawRect(x, y, w, h)</code>     | draw rectangle $[x \dots x + w, y \dots y + h]$ |
| <code>drawOval(x, y, w, h)</code>     | draw oval $[x \dots x + w, y \dots y + h]$      |
| <code>drawLine(x1, y1, x2, y2)</code> | draw line from $(x1, y1)$ to $(x2, y2)$         |
| <code>drawString(s, x, y)</code>      | draw string $s$ at $(x, y)$                     |
| <code>fillRect(x, y, w, h)</code>     | fill rectangle $[x \dots x + w, y \dots y + h]$ |
| <code>fillOval(x, y, w, h)</code>     | fill oval $[x \dots x + w, y \dots y + h]$      |
| <code>setColor(c)</code>              | set drawing color to $c$                        |

```
Drawing.graphics.fillRect(60, 80, 225, 30);
```

- Draw a rectangle whose left upper corner is 60 pixels to the right and 80 pixels down and that extends 225 pixels to the right and 30 pixels down.

```
Drawing.graphics.setColor(Color.red);
```

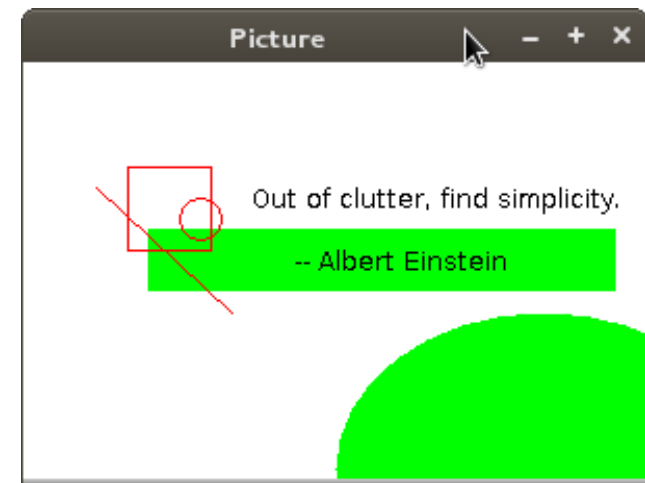
- Set the drawing color of the following operations to red.

## Example

```
public static void main(String[] args) {
 Drawing.begin("Picture", 300, 200);

 Drawing.graphics.setColor(Color.green);
 Drawing.graphics.fillRect (60, 80, 225, 30);
 Drawing.graphics.fillOval (150, 120, 200, 150);
 Drawing.graphics.setColor(Color.red);
 Drawing.graphics.drawRect (50, 50, 40, 40);
 Drawing.graphics.drawOval (75, 65, 20, 20);
 Drawing.graphics.drawLine (35, 60, 100, 120);
 Drawing.graphics.setColor(Color.black);
 Drawing.graphics.drawString ("Out of clutter, find simplicity.", 110, 70);
 Drawing.graphics.drawString ("-- Albert Einstein", 130, 100);

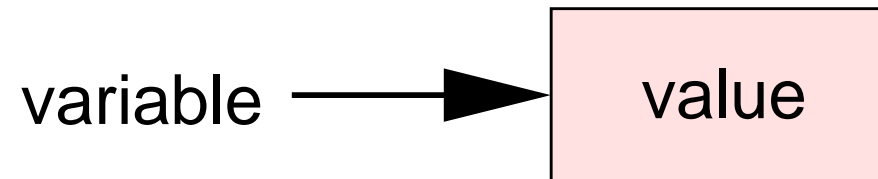
 Drawing.end();
}
```





## Variables

- Program data are kept in **variables**:
  - Name for a memory address at which a data value is stored.
  - Container/box that contains values.



- Variable declaration.
  - Creation of a variable and initialization with a value.
- Variable assignments.
  - Modification of the variable content.

## Variable Declarations

- Variable declaration:

1. reserve a portion of memory space to hold a particular type of value,
2. to initialize this portion with a particular value of this type,
3. refer further on to this portion by a particular name.

- Statements:

```
String name = "Markus";
int age = 21;
System.out.println(name + " is " + age + " years old.");
```

- Variable *name* of type `String` initialized as “Markus”.
- Variable *age* of type `int` initialized as 21.

- Output:

```
Markus is 21 years old.
```

## Variable Declarations

- General Format:

*type identifier = value ;*

- *type* denotes the kind of data that the variable may hold, e.g. `int` for integer numbers and `String` for character strings;
- *identifier* is the name of the variable which may be used from the point of the declaration to the end of the method;
- *value* is the initial value of the variable.

- Naming convention:

- Variable names usually start with a lower-case letter.
- `i`, `number`, `temperature`.

It is an error to declare a variable twice (in the same scope).

## Assignments

- **Assignment** statement:

*identifier = value;*

– “*identifier* becomes *value*”.

– Write the value denoted by the right hand side to the variable identified by the left hand side.

- **Program:**

```
int x = 3;
```

```
x = x+1;
```

– Right: “*x* becomes *x+1*”.

– Wrong: “*x* is *x+1*”.

– Old value: 3; new value: 4.

“Varying” means “changing”.

## Example

- Statements:

```
String name = "Markus";
int age = 21;
System.out.println(name + " is " + age + " years old.");
name = "Michaela";
age = 23;
System.out.println(name + " is " + age + " years old.");
```

- Output:

```
Markus is 21 years old.
Michaela is 23 years old.
```

Effect of a statement depends on the values of the variables used.

## Assignments

- Value in assignment must be of declared variable type.

- Program:

```
int age = 21;
age = "twentyone";
```

- Compiler:

```
Main.java:6: incompatible types
found : java.lang.String
required: int
 age = "twentyone";
 ^
```

- May omit initialization, if assigned before first use:

```
int age;
age = 21;
System.out.println(age);
```

## Constant Declarations

- **Constant** Declaration:

```
final type identifier = value;
```

- *identifier* is declared as “constant”.
- Value cannot be changed any more (no assignment possible).

- Use constant declarations for special values:

```
final int MAX_VALUE = 127;
final String SYS_DIR = "c:\\windows";
```

- Names of constants often written in capital letters to distinguish them from variables.

“Constant” means “not changing”.

## Choosing Variable Names

- Name should be a noun denoting the value it holds:

```
int temperature = 17;
int maximumTemperature = 17;
String directoryName = "c:\\windows";
```

- Names should better not be too long:

```
int maxTemp = 17;
String dirName = "c:\\windows";
```

- Names should not be too short:

```
int mt = 17;
String dn = "c:\\windows";
```

- Annotate declarations by comments:

```
int temp = 0; // temperature taken in last measurement
int maxTemp = 0; // maximum temperature of all measurements
int avgTemp = 0; // average temperature of all measurements
```