



Bogomil Dimitrov
Technical Trainer

Software University
<http://softuni.bg>



Loops, Methods, Classes

Using Loops, Defining and Using
Methods, Using API Classes,
Exceptions, Defining Classes



A complex, abstract visualization of Java code, rendered in yellow against a dark background. The code is presented in a non-linear, branching structure, resembling a neural network or a fractal pattern. Key words like "private", "public", "try", "catch", and "Override" are visible, along with snippets of class definitions and method bodies. A large, glowing yellow "Java" logo is positioned in the center of the visualization.

```
private final String[] studys = {"TodSofts", "Name", "Apensoft"};
private DefaultTableModel dtm = new DefaultTableModel();
private JTable personTable = new JTable(dtm);
private BDManagement bdm = new BDManagement();

private JMenuBar menuBar;
private JMenu actionMenu;
private JMenuItem addMenuItem, removeMenuItem, readMenuItem;
private JScrollPane scrollPane;

private List<Integer> ids = new ArrayList<Integer>();

public GUI() {
    setLayout("JFrame CON GUI");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setLayout(null);
    this.setSize(800, 600);
    this.addContentPanel();
    this.setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    // TODO Auto-generated method stub
    if(e.getSource()==addMenuItem){
        try {
            Class.forName("com.mysql.jdbc.Driver");
            db = DriverManager.getConnection(db,userAndpass,userAndpass);
            st = ct.createStatement();
            System.out.println("ok");
            JOptionPane.showMessageDialog(null, "Success!");
            System.out.println("error");
        } catch (Exception e1) {
            System.out.println("error");
        }
    }
}

public Statement getStatement() {
    return st;
}

public Statement getStatement() {
    return st;
}

@Override
public void keyPressed(KeyEvent e) {
    // TODO Auto-generated method stub
    if(e.getKeyCode()==KeyEvent.VK_ENTER){
        String lines = textto.getText();
        int numbs = Integer.parseInt(lines.substring(lines.indexOf("-")));
        int total = numbs - 1;
        String res = lines.substring("1", total);
        res = res.substring(0, res.length() - 1);
        res = res.substring(res.indexOf("1"));
        res = res.substring(res.indexOf("1"));
    }
}
```

Table of Contents

1. Loops

- while, do-while, for, for-each

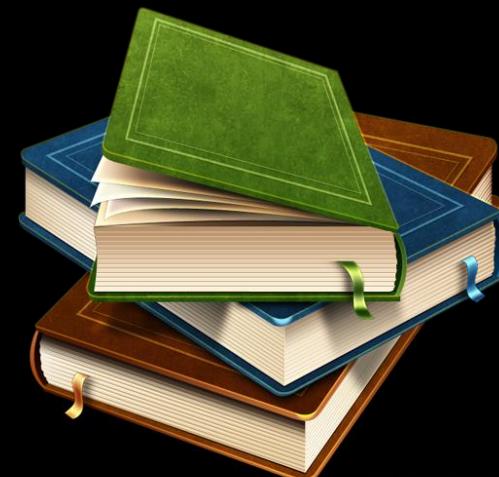
2. Methods

- Defining Methods
- Invoking Methods

3. Using the Java API Classes

4. Exception Handling Basics

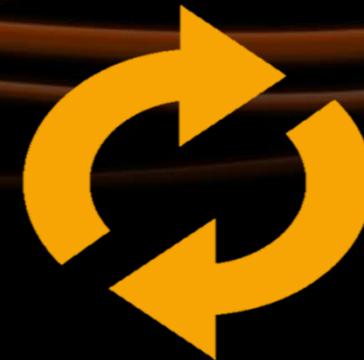
5. Defining Simple Classes



Warning: Not for Absolute Beginners

- The "**Java Basics**" course is NOT for absolute beginners
 - Take the "C# Basics" course at SoftUni first:
<https://softuni.bg/courses/csharp-basics>
 - The course is for beginners, but with previous coding skills
- Requirements
 - Coding skills – entry level
 - Computer English – entry level
 - Logical thinking





Loops

Loop: Definition

- A **loop** is a control statement that repeats the execution of a block of statements

```
while (condition) {  
    statements;  
}
```



- May execute a code block fixed number of times
- May execute a code block while given condition holds
- May execute a code block for each member of a collection
- Loops that never end are called an **infinite loops**

While Loop

- The simplest and most frequently used loop

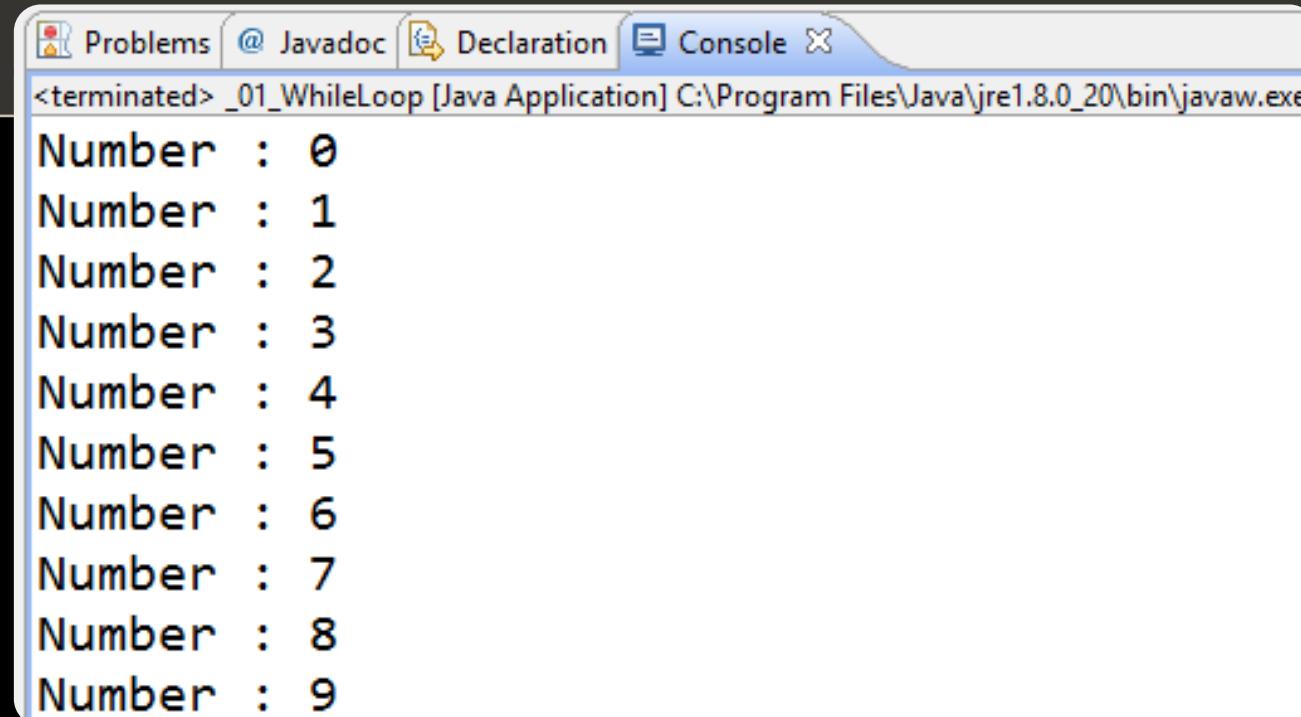
```
while (condition) {  
    statements;  
}
```



- The repeat condition
 - Returns a boolean result of **true** or **false**
 - Also called loop condition

While Loop – Example: Numbers 0...9

```
int counter = 0;  
while (counter < 10) {  
    System.out.printf("Number : %d\n", counter);  
    counter++;  
}
```



The screenshot shows a Java application window with tabs for Problems, Javadoc, Declaration, and Console. The title bar indicates the application is named '_01_WhileLoop' and is a Java Application. The console tab displays the following output:

```
Number : 0  
Number : 1  
Number : 2  
Number : 3  
Number : 4  
Number : 5  
Number : 6  
Number : 7  
Number : 8  
Number : 9
```

Do-While Loop

- Another classical loop structure is:

```
do {  
    statements;  
}  
while (condition);
```



- The block of **statements** is repeated
 - While the boolean loop **condition** holds
- The loop is executed at least once

Product of Numbers [N..M] – Example

- Calculating the product of all numbers in the interval [n..m]:

```
Scanner input = new Scanner(System.in);
int n = input.nextInt();
int m = input.nextInt();
int number = n;
BigInteger product = BigInteger.ONE;
do {
    BigInteger numberBig = new BigInteger("+" + number);
    product = product.multiply(numberBig);
    number++;
}
while (number <= m);
System.out.printf("product[%d..%d] = %d\n", n, m, product);
```

For Loops

- The classical **for**-loop syntax is:

```
for (initialization; test; update) {  
    statements;  
}
```



- Consists of
 - Initialization statement
 - Boolean **test** expression
 - Update statement
 - Loop **body** block

For Loop – Examples

- A simple **for**-loop to print the numbers 0...9:

```
for (int number = 0; number < 10; number++) {  
    System.out.print(number + " ");  
}
```

- A simple **for**-loop to calculate n!:

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

Using the `continue` Operator

- `continue` bypasses the iteration of the inner-most loop
- Example: sum all odd numbers in [1...n], not divisors of 7:



```
int n = 100;
int sum = 0;
for (int i = 1; i <= n; i += 2) {
    if (i % 7 == 0) {
        continue;
    }
    sum += i;
}
System.out.println("sum = " + sum);
```

Using the **break** Operator

- The **break** operator exits the inner-most loop

```
public static void main(String[] args) {  
    int n = new Scanner(System.in).nextInt();  
    // Calculate n! = 1 * 2 * ... * n  
    int result = 1;  
    while (true) {  
        if (n == 1)  
            break;  
        result *= n;  
        n--;  
    }  
    System.out.println("n! = " + result);  
}
```



For-Each Loop

- The typical **for-each** loop syntax is:

```
for (Type element : collection) {  
    statements;  
}
```



- Iterates over all the elements of a collection
 - The **element** is the loop variable that takes sequentially all collection values
 - The **collection** can be list, array or other group of elements of the same type

For-Each Loop – Example

- Example of **for-each** loop:

```
String[] days = { "Monday", "Tuesday", "Wednesday",
                  "Thursday", "Friday", "Saturday", "Sunday" };
for (String day : days) {
    System.out.println(day);
}
```

- The loop iterates over the array of day names
 - The variable **day** takes all its values
- Applicable for all collections: arrays, lists, strings, etc.

Nested Loops

- Loops can be nested (one inside another)
- Example: print all combinations from TOTO 6/49 lottery

```
for (int i1 = 1; i1 <= 44; i1++)  
    for (int i2 = i1 + 1; i2 <= 45; i2++)  
        for (int i3 = i2 + 1; i3 <= 46; i3++)  
            for (int i4 = i3 + 1; i4 <= 47; i4++)  
                for (int i5 = i4 + 1; i5 <= 48; i5++)  
                    for (int i6 = i5 + 1; i6 <= 49; i6++)  
                        System.out.printf("%d %d %d %d %d %d\n",  
                            i1, i2, i3, i4, i5, i6);
```



Loops

Live Demo

```
private static void printAsterix(int count) {  
    for (int i = 0; i < count; i++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}  
  
public static void main(String[] args) {  
    int n = 5;  
    for (int i = 1; i <= n; i++) {  
        printAsterix(i); ——————  
    }  
}
```

Methods

Defining and Invoking Methods

Methods: Defining and Invoking

- Methods are named pieces of code
 - Defined in the class body
 - Can be invoked multiple times
 - Can take parameters
 - Can return a value

```
private static void printAsterix(int count) {  
    for (int i = 0; i < count; i++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}  
  
public static void main(String[] args) {  
    int n = 5;  
    for (int i = 1; i <= n; i++) {  
        printAsterix(i); ——————  
    }  
}
```

Methods with Parameters and Return Value

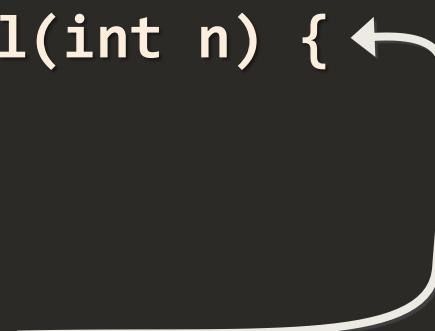
```
static double calcTriangleArea(double width, double height) {  
    return width * height / 2;  
}  
  
public static void main(String[] args) {  
    Scanner input = new Scanner(System.in);  
    System.out.print("Enter triangle width: ");  
    double width = input.nextDouble();  
    System.out.print("Enter triangle height: ");  
    double height = input.nextDouble();  
    System.out.println("Area = " + calcTriangleArea(width, height));  
}
```

Method names in Java
should be in **camelCase**

Recursion

- Recursion == method can calls itself

```
public static void main(String[] args) {  
    int n = 5;  
    long factorial = calcFactorial(n);  
    System.out.printf("%d! = %d", n, factorial);  
}  
  
private static long calcFactorial(int n) { ←  
    if (n <= 1) {  
        return 1;  
    }  
    return n * calcFactorial(n-1); ——————  
}
```



Method Return Types

- Type **void**
 - Does not return a value directly by itself

```
static void addOne(int n) {  
    n += 1;  
    System.out.println(n);  
}
```

- Other types
 - Return values, based on the **return type** of the method

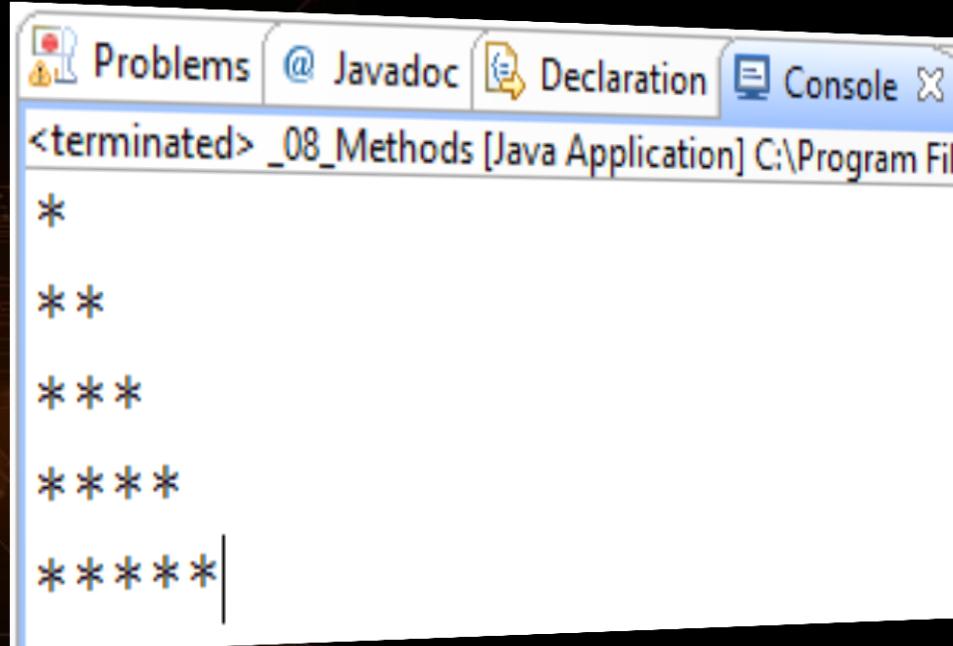
```
static int plusOne(int n) {  
    return n + 1;  
}
```

Method Access Modifiers

- **private**
 - Accessible only inside the current class. No subclass can call this
- **package** (default)
 - Accessible only inside the package. Subclasses can call this
- **protected**
 - Accessible by subclasses even outside the current package
- **public**
 - All code can access this, e.g. external classes

Methods

Live Demo



The screenshot shows a Java application running in an IDE. The title bar reads "Problems @ Javadoc Declaration Console <terminated> _08_Methods [Java Application] C:\Program File". The console tab is active, displaying the following output:
*
**

*****|

```
private static void printAsterix(int count) {  
    for (int i = 0; i < count; i++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}  
  
public static void main(String[] args) {  
    int n = 5;  
    for (int i = 1; i <= n; i++) {  
        printAsterix(i); ——————  
    }  
}
```



Using the Java API Classes

Build-in Classes in the Java API

- Java SE provides thousands of ready-to-use classes
 - Located in packages like **java.lang**, **java.math**, **java.net**, **java.io**, **java.util**, **java.util.zip**, etc.
- Using static Java classes:

```
LocalDate today = LocalDate.now();
double cosine = Math.cos(Math.PI);
```

- Using non-static Java classes

```
Random rnd = new Random();
int randomNumber = 1 + rnd.nextInt(100);
```



Using the Java API Classes

Live Demo



Exception Handling Basics

Catch and Throw Exceptions

Handling Exceptions

- In Java exceptions are handled by the **try-catch-finally** construction

```
try {  
    // Do some work that can raise an exception  
} catch (SomeException ex) {  
    // Handle the caught exception  
} finally {  
    // This code will always execute  
}
```



- catch** blocks can be used multiple times to process different exception types

Handling Exceptions – Example



```
public static void main(String[] args) {  
    String str = new Scanner(System.in).nextLine();  
    try {  
        int i = Integer.parseInt(str);  
        System.out.printf(  
            "You entered a valid integer number %d.\n", i);  
    }  
    catch (NumberFormatException nfex) {  
        System.out.println("Invalid integer number: " + nfex);  
    }  
}
```

The "throws ..." Declaration

- A method in Java could declare "**throws SomeException**"
 - This says "I don't care about **SomeException**", please re-throw it

```
public static void copyStream(InputStream inputStream,
    OutputStream outputStream) throws IOException {
    byte[] buf = new byte[4096]; // 4 KB buffer size
    while (true) {
        int bytesRead = inputStream.read(buf);
        if (bytesRead == -1)
            break;
        outputStream.write(buf, 0, bytesRead);
    }
}
```

Resource Management in Java

- When we use a resource that is expected to be closed, we use the `try-with-resources` statement

```
try(  
    BufferedReader fileReader = new BufferedReader(  
        new FileReader("somefile.txt"));  
) {  
    while (true) {  
        String line = fileReader.readLine();  
        if (line == null) break;  
        System.out.println(line);  
    } catch (IOException ioex) {  
        System.err.println("Cannot read the file .");  
    }  
}
```

```
class Point {  
    private double x, y;  
  
    public Point(double x, double y) {}  
  
    public double getX() {}  
  
    public void setX(int x) {}  
  
    public double getY() {}  
  
    public void setY(double y) {}  
  
    public double calcDistance(Point otherPoint) {}  
  
    public static Point findMidPoint(Point p1, Point p2) {}  
}
```

Defining Simple Classes

Using Classes to Hold a Set of Fields

Defining Classes in Java

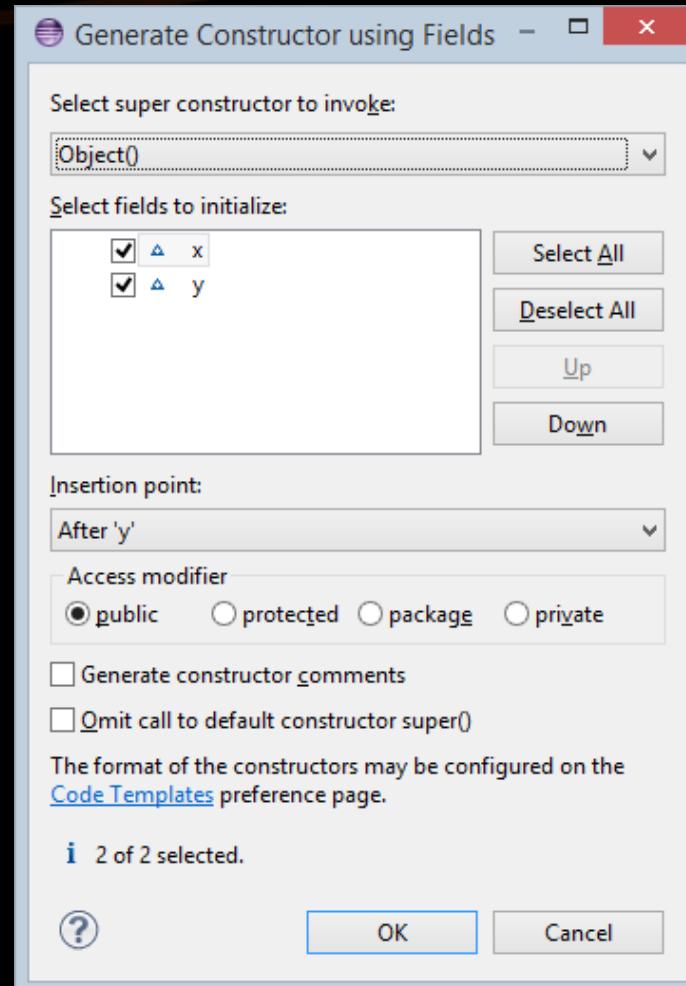
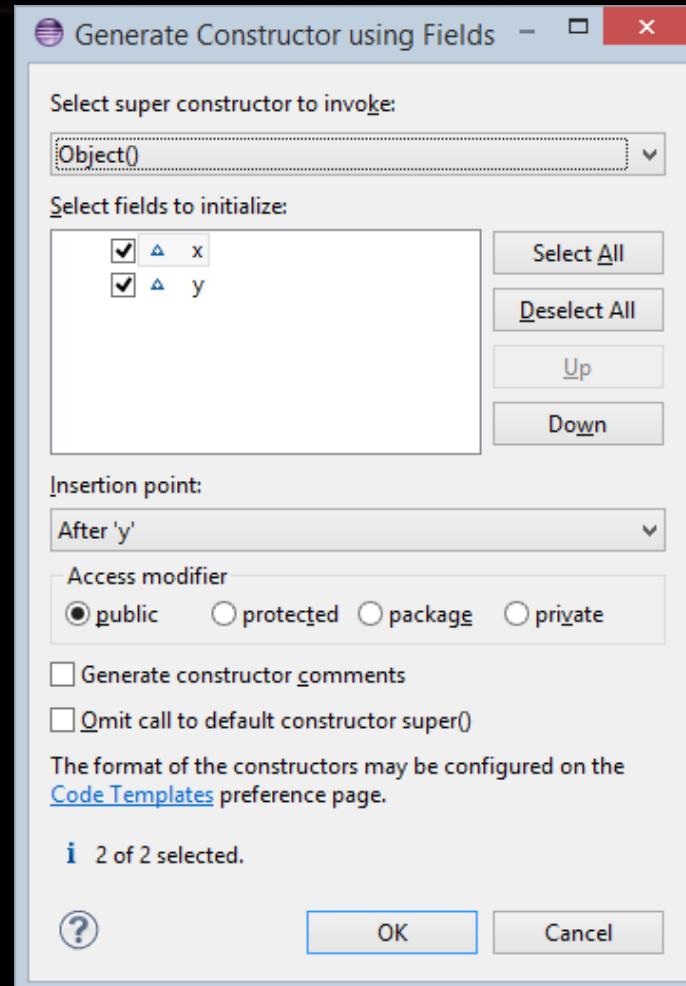
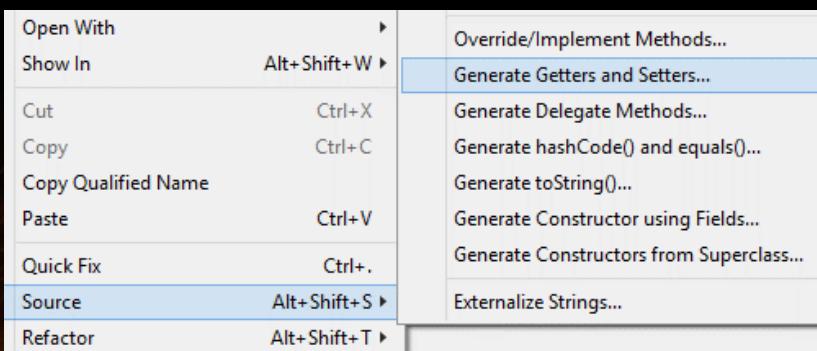
- Classes in Java combine a set of named fields / properties
- Defining a class **Point** holding **X** and **Y** coordinates:

```
class Point {  
    private int x, y;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int getX() {  
        return x;  
    }
```

```
        public void setX(int x) {  
            this.x = x;  
        }  
  
        public int getY() { ... }  
        public void setY(int y) { ... }  
    }
```

Defining Classes in Eclipse

- Eclipse provides tools for automatic generation of constructors and getters / setters for the class fields



```
class Point {  
    private double x, y;  
  
    public Point(double x, double y) {}  
  
    public double getX() {}  
  
    public void setX(int x) {}  
  
    public double getY() {}  
  
    public void setY(double y) {}  
  
    public double calcDistance(Point otherPoint) {}  
  
    public static Point findMidPoint(Point p1, Point p2) {}  
}
```

Defining Simple Classes

Live Demo

Summary

- Java supports the classical loop constructs
 - **while, do-while, for, for-each**
 - Similarly to C#, JavaScript, PHP, C, C++, ...
- Java support **methods**
 - Methods are named code blocks
 - Can take parameters and return a result
- Java supports classical **exception handling**
 - Through the **try-catch-finally** construct
- Developers can define their own **classes**
 - With fields, methods, constructors, getters, setters, etc.



Java Basics – Loops, Methods, Classes

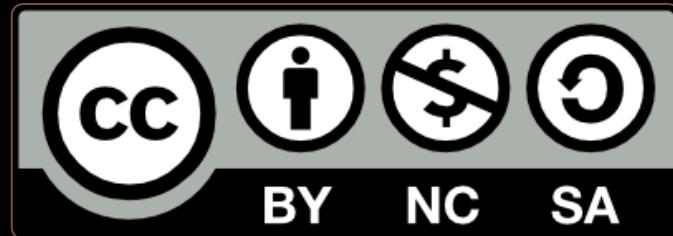


Questions?



License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license



- Attribution: this work may contain portions from
 - "Fundamentals of Computer Programming with Java" book by Svetlin Nakov & Co. under CC-BY-SA license
 - "C# Basics" course by Software University under CC-BY-NC-SA license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University @ YouTube
 - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg

