1. Class Relationships

See also:

- Class declarations define new reference types and describe how they are implemented.
- Constructors are similar to methods, but cannot be invoked directly by a method call; they are used to initialize new class instances. Like methods, they may be overloaded.
- Static initializers are blocks of executable code that may be used to help initialize a class when it is first loaded.
- The body of a class declares members, static initializers, and constructors.
- The scope of the name of a member is the entire declaration of the class to which the member belongs. Field, method, and constructor declarations may include the access modifiers public, protected, or private. The members of a class include both declared and inherited members.
- Newly declared fields can hide fields declared in a superclass or superinterface. Newly declared methods can hide, implement, or override methods declared in a superclass.
- Visibility modifier: public/private/protected
- Return type: void/primitive type/reference to an object
- Class methods/Class variables are declared with static.
- Static declaration inside a method change the lifetime of a variable.
1.1. Example

```java
1 public class X {
2     static int cI = 0;
3     int oI = 1;
4
5     public X() {
6 }
7     static public void cm(int i){
8         cI = i;
9         System.out.println("cm: " + cI);
10     }
11     public void om(int i){
12         cI = i;
13         oI = i;
14         System.out.println("om: " + this);
15         System.out.println(" cI " + cI);
16         System.out.println(" oI " + oI);
17     }
18
19     public static void main(String args[]) {
20         X aX = new X();
21         X aaX = new X();
22
23         aX.cm(1);
24         aaX.cm(2);
25
26         aX.om(3);
27         aaX.om(4);
28     }
29 }
```

Source Code: Src/5/X.java

```bash
% java X
  cm: 1
  cm: 2
  om: X@1ad086a
    cI 3
    oI 3
  om: X@10385c1
    cI 4
    oI 4
```

1.2. Class Details

1.3. Static in Classes/Methods Lifetime

- Class Variables: If a variable is declared static, there exists exactly one incarnation of the field,
- Static Methods: A method that is declared static is called a class method. A class method is always invoked without reference to a particular object.
• Non Static Methods: A method that is not declared static is called an instance method, and sometimes called a non-static method. An instance method is always invoked with respect to an object, which becomes the current object to which the keywords this and super refer during execution of the method body.

• Variables can be declared:
  — static: class variable
  — final: can be assigned once or zero time
  — transient: not stored or saved via the standard serialization process
  — volatile: A variable may be modified by multiple threads. This gives a hint to the compiler to fetch the value each time, rather store a locale copy. This also prohibits same optimization procedures.

• See also:

```java
public class Overview {
    int instanceVariable;
    static int classVariable;
    final int finalVariable; // static?
    volatile int volatileVariable;
    transient int transientVariable;

    public Overview() {
        finalVariable = 42;
    }
    public Overview(int aLocalVariable) {
        finalVariable = 43;
    }
    void instanceMethod() {
        finalVariable = 43;
        instanceVariable = 22;
        classVariable = 33;
    }
    static void classMethod() {
        classVariable = 3;
    }

    public static void main(String args[]) {
        Overview aOverview = new Overview();
        Overview bOverview = new Overview();
        Overview cOverview = new Overview(1);
        aOverview.instanceMethod();
        instanceMethod();
        bOverview.classMethod();
        // values of aOverview.? bOverview.?
        // aOverview.finalVariable??
    }
}
```

Source Code: Src/5/Overview.java
1.4. Parameter Passing

- The formal parameters of a method, if any, are specified by a list of comma-separated parameter specifiers.
- Each parameter specifier consists of a type and an identifier (optionally followed by brackets) that specifies the name of the parameter.
- If a method has no parameters, only an empty pair of parentheses appears in the method’s declaration.
- If two formal parameters are declared to have the same name (that is, their declarations mention the same identifier), then a compile-time error occurs.
- When the method is invoked, the values of the actual argument expressions initialize newly created parameter variables, each of the declared type, before execution of the body of the method.
- The scope of formal parameter names is the entire body of the method. These parameter names may not be redeclared as local variables or exception parameters within the method; that is, hiding the name of a parameter is not permitted.
- call by value
1.5. Example I

```java
public class ExampleClass {
    int aLocalVariable = 3;

    public ExampleClass() {
        aLocalVariable = 2;
    }

    public ExampleClass(int aLocalVariable) {
        this.aLocalVariable = aLocalVariable;
        aLocalVariable = 6;
    }

    public static void main(String args[]) {
        ExampleClass aExampleClass = new ExampleClass();
        aExampleClass = new ExampleClass(3);
        System.out.println("the value is: "+ aExampleClass.aLocalVariable);
        System.out.println(aLocalVariable);
    }
}
```

Source Code: Src/5/ExampleClass.java

Questions:
- How does the JVM find the main method, when you execute java ExampleClass?
- Describe the execution order of the constructors.
- Which variables will be modified when?

1.6. Example II

```java
public class ExampleClassII {

    static int aInstanceVariable = 42;
    static int[] intArray = new int[4];

    public ExampleClassII(int aInstanceVariable) {
        aInstanceVariable = aInstanceVariable;
    }

    public static void methodInt(int aInt) {
        aInt = 33;
    }

    public static void methodIntArray(int array[]) {
        array[0] = 42;
    }

    public static void main(String args[]) {
        ExampleClassII aExampleClassII = new ExampleClassII(3);
        methodInt(aInstanceVariable);
        methodIntArray(intArray);
        System.out.println("aInstanceVariable = " + aInstanceVariable);
        System.out.println("intArray[0] = " + intArray[0]);
    }
}
```
23 }  

Source Code: Src/5/ExampleClassII.java

Questions:
- How does the JVM find the variable aStaticLocalVariable?
- Which variables will be modified when?
1.7. Example III

```java
public class ExampleClassIII {
    String aString = null;

    public void method(String a) {
        a = new String("set in method");
        System.out.println("2. method:a:" + a);
    }

    public void test() {
        String aString = new String("set in test");

        System.out.println("1. test:aString:" + aString);
        method(aString);
        System.out.println("3. test:aString:" + aString);
    }

    public static void main(String args[]) {
        new ExampleClassIII().test();
    }
}
```

Source Code: Src/5/ExampleClassIII.java

1.8. Example IV

```java
public class XX {
    int instanceV = 1;

    static XX bbbbbb;
    static XX aaaaaa;

    public XX() {
    }

    public void method(int i){
        instanceV = i;
    }

    public String toString() {
        return "instanceV = " + instanceV;
    }

    public void m2(int i){
        aaaaaa.method(-9);
        method(12);
        System.out.println("-----------------------------------------");
        System.out.println("print itself : " + this);
        System.out.println("print aaaaaa: " + aaaaaa);
        System.out.println("-----------------------------------------");
    }
}
```
```java
public static void main(String args[]) {
    bbbbb = new XX();
    aaaaaa = new XX();
    bbbbb.m2(3);
    aaaaaa.m2(24);
}
```
1.9. Example VI

    public class XXX {
        int oI = 1;
        XXX aXXX = new XXX();
        public XXX() {
        }
        public XXX(XXX aXXX) {
            this.aXXX = aXXX;
        }
        public void m1(int i) {
            oI = i;
        }
        public void m2(int i) {
            oI = i;
            m1(12);
            aXXX.m1(24);
            System.out.println("om: " + this);
            System.out.println("oI " + oI);
            System.out.println("aXXX.om: " + aXXX);
            System.out.println("aXXX.oI " + aXXX.oI);
        }
        public static void main(String args[]) {
            XXX aaXXX = new XXX(new XXX());
            aaXXX.m2(3);
        }
    }

Source Code: Src/5/XXX.java

- does it compile? ja. Draw the memory pic during execution
- Does it execute? Ja, aber es wird sterben, wie im C. ein wort fehlt Output?

1.10. A Point Class

- An example without any comment:
  
  I. Use of a Point Class:
public class TestPoint {
    private static Point aPoint;

    /**
     * The main program.
     * @param args command line arguments (ignored)
     */
    public static void main(String args[]) {
        System.out.println("Point.soManyPoints = " + Point.soManyPoints() );
        System.out.println("x = " + aPoint.getX() );
        System.out.println("y = " + aPoint.getY() );

        aPoint = new Point();
        aPoint.initPoint(4, 5);
        System.out.println("x = " + aPoint.getX() );
        System.out.println("y = " + aPoint.getY() );

        aPoint.move(6, 7);
        System.out.println("x = " + aPoint.getX() );
        System.out.println("y = " + aPoint.getY() );

        System.out.println("nPoints = " + aPoint.getNPoints() );
        System.out.println("aPoint.soManyPoints = " + aPoint.soManyPoints() );
    }
}

Source Code: Src/5/TestPoint.java
II. The *Point* Class:

```java
/**
 * This class implements a point in a two dimensional area.
 * All method print when they are called.
 *
 * @version $Id$
 * @author hp bischof
 *
 * Revisions:
 * $Log$
 */

public class Point {
    // class variable
    static int nPoints; // so many points were created.

    private int x; // x coordinate of the point
    private int y; // y coordinate of the point

    /**
     * Default Constructor.
     * Increases the counter nPoints by 1.
     *
     * @return Point a Point object
     */
    public Point(){
        super();
        System.out.println(" in Point() constructor");
        nPoints ++;
    }

    /**
     * Constructor.
     * initialize x and y values of a point
     *
     * @param x x coordinate
     * @param y y coordinate
     *
     * @return Point a Point object
     */
    public Point(int x, int y){
        super();
        int i ++;
        this.x = x;
        this.y = y;
        System.out.println(" in Point(int, int) constructor");
    }

    /**
     * So many points have been created.
     */
```
@return int So many points have been created

public static int soManyPoints(){
    return nPoints;
}

/**
 * initializes x and y of a point.
 *
 * @param x int x coordinate
 * @param y int y coordinate
 *
 * @return Point a Point object
 */
public Point initPoint(int x, int y){
    System.out.println(" in initPoint(int, int)");
    this.x = x;
    this.y = y;
    return this;
}

/**
 * move a point
 *
 * @param x int delta x value
 * @param y int delta y value
 *
 * @return Point a Point object
 */
public Point move(int x, int y){
    System.out.println(" in move(int, int)");
    this.x += x;
    this.y += y;
    return this;
}

/**
 * Returns the x coordinate of a point
 *
 * @return int x value
 */
public int getX(){
    System.out.println(" in getX()");
    return this.x;
}

/**
 * Returns the y coordinate of a point
 *
public int getY()
{
    System.out.println(" in getY()");
    return this.y;
}

/**
 * Returns how many points are created so far.
 *
 * @return int nPoints
 */

public int getNPoints()
{
    System.out.println(" in getNPoints()");
    return this.nPoints;
}

Source Code: Src/5/Point.java

II. Execution of the test program:

Point.soManyPoints = 0
    in Point() constructor
    in Point(int, int) constructor
    in getX()
    x = 2
    in getY()
    y = 3
    in Point() constructor
    in initPoint(int, int)
    in getX()
    x = 4
    in getY()
    y = 5
    in move(int, int)
    in getX()
    x = 10
    in getY()
    y = 12
    in getNPoints()
    nPoints = 2
    aPoint.soManyPoints = 2

You may find the javadoc pages
1.11. Additional Examples

See

```java
public class Scope_1 {
    String aString = null;

    public void method(String aString) {
        this.aString = new String("set in method");
        System.out.println("2. method:aString:" + this.aString);
    }

    public void test() {
        String aString = new String("set in test");
        System.out.println("1. test:aString:" + aString);
        method(aString); // is there a way that "set in method"
        System.out.println("3. test:aString:" + aString);
    }

    public static void main(String args[]) {
        new Scope_1().test();
    }
}
```

Source Code: Src/5/Scope_1.java

```java
public class Scope_2 {
    String aString = null;

    public void test() {
        String aString = new String("set in test");
        // if ( true )
        {
            String aString = new String("set in test");
        }
        System.out.println("1. test:aString:" + aString);
        method(aString); // is there a way that "set in method"
        System.out.println("3. test:aString:" + aString);
    }

    public static void main(String args[]) {
        new Scope_2().test();
    }
}
```

Source Code: Src/5/Scope_2.java

```java
public class Scope_3 {
    String aString = null;

    public void test() {
        int i;
```
class Test {
    int i;
}
for (int index = 0; index < 10; index ++ ) {
    System.out.println("index = " + index );
}
public static void main(String args[] ) {
    new Scope_3().test();
}

public class Scope_4 {
    String aString = null;
    public static void test_2() {
        int k = 0;
        for (int index = 0; index < 10 ; index ++ ) {
            int k = 3;
        }
    }
    public static void test() {
        int i;
        int k = 0;
        switch (k) {
            case 1: {
                int i=1;
                System.out.println("1: i == " + i);
            }
            break;
            default:
                System.out.println("something went wrong!");
                break;
        }
    }
    public static void main(String args[] ) {
        test();
    }
}

// see http://docs.oracle.com/javase/specs/jls/se7/html/jls-6.html#jls-6.3 example 6.4-1

1.12. Question arrived 22:44

Newly declared fields can hide fields declared in a superclass or superinterface. Newly declared
methods can hide, implement, or override methods declared in a superclass. Would hiding just mean that new declared things would be accessed first, because we still can access the superclass members by means of (super.field) and/or (super.method(arg))?

•

Static declaration inside a method change the lifetime of a variable. Can static variables be declared inside of a method? How does the JVM find the variable aStaticLocalVariable?(Question from the second link)

•

System.out.println("om: " + this); What kind of address is this? -> om: X@1ad086a Is it the value of the reference to this object? Is it some virtual address provided by the JVM?

•

What are transient and volatile variables? When are they used? If I assign a value to the final variable at the time of declaration but I don’t declare it to be static, will still there be copies of this variable with the objects or this variable will behave like a static final?

•

This program is confusing for me. I am not able to comprehend the concept that it introduces.

•

Some confusions regarding scopes.

void func(){
    int index=2;
    for (int index=0; index<5; index++) { //some code}
}

It doesn’t compile saying that the variable "index" is already defined.

void func1(){
    for(int index=0; index<5; index++) { //some code}

    System.out.println(index); // Does not compile saying that the variable "index" cannot
    // be accessed from here.
}

Similar thing happens in switch case. I also read somewhere that hiding of parameter inside the method body is illegal. What happens in the case of inner classes?

•

s.findInLine("(\d+) fish (\d+) fish (\w+) fish (\w+)");
Which of the following method executes it?
StringfindInLine(P attern pattern) OR StringfindInLine(String pattern)
Is it a regular expression? If yes, why is the argument using the delimiter to be " fish ". It should have taken the entire thing as one.

•

How does the auto-boxing actually proceeds in the following program from this link:-
public class SimpleBoxingI {
    public static void main( String[] args ) {
        Integer n1 = new Integer( 42 );
        int i1 = n1;
        int i2 = 31;
        Integer n2 = i1 + i2 + n1; // are all of them boxed?
        System.out.println(n1 + " " + n2 + " " + i1 + " " + i2);
    }
}

One more question regarding boxing:-
public class StringToInt {
    public static void main(String[] args) {
        int i;
        Integer aInt = new Integer("4");

        i = aInt;
        // Will the difference between the line above and the line below be auto unboxing and unboxing done by a method respectively?
        i = aInt.intValue();
        i = Integer.parseInt("4");
    }
}

public class SimpleBoxingTypes {
    public static void main(String[] args) {
        Float f1 = new Float(42F);
        Integer i1 = new Integer(42);
        Double d1 = new Double(42.0);

        double f = f1 + (double)i1 + (double)d1;
        // float ff = f1 + (float)i1 + ((float)d1);

        System.out.println(f);
    }
}

Boxing - primitive to Object type
Unboxing - Object to primitive type
What is the concept I am missing in this example? I see only type casting.
Secondly, why does the commented float line doesn’t compile? Why does it start compiling when I use the primitive types instead of object types? Why do the boxing and unboxing things don’t help? Does this demonstrate some type of a difference between the two types?

• I see a break statement at the end of the default case everywhere? Does anything significantly change if I don’t provide it.

•

Each parameter specifier consists of a type and an identifier (optionally followed by brackets) that specifies the name of the parameter.
Where can we use brackets?