Software Engineering and Programming Basics

Heap, Stack, Parameter, Cloning

Authors of slides:
Prof. Dr.-Ing. Janet Siegmund
Prof. Dr.-Ing. Norbert Siegmund
Prof. Christian Lengauer
Partly extracted from script of PD Dr. Christian Bachmaier

- Implement a method that returns how often a character is contained in a string

Helpful methods of the class String:
- int length(): Returns the length of this string.
- char charAt(int index)
Returns the char value at the specified index.

- Proceed step by step
  1. What is the method head? (parameter, return type)?
  2. What does the method body look like?
     - Iteration over characters of a string with a loop… which loop?
     - Compare current character with a given character… how?
     - Increment the number of characters… how?
     - Return the sum… how?

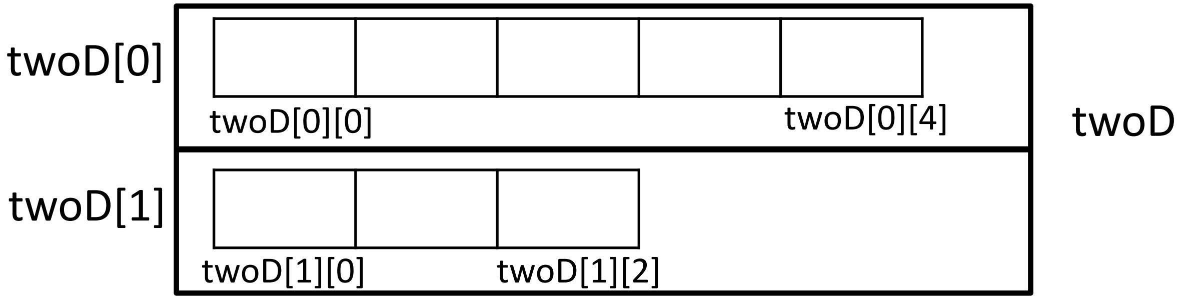
5 to 7 minutes

Catching Up I: Multidimensional Arrays

- Arrays are stored in arrays (e.g. matrix)
- Declaration via additional „[ ]“

```
int [][] twoD = new int[2][];
twoD[0] = new int[5];
twoD[1] = new int[3];
```

Here, it is possible without defining the size, because not all rows need to have the same length



twoD[0]

twoD[0][0]

twoD[0][4]

twoD[1]

twoD[1][0]

twoD[1][2]

twoD

```
int [][] uniform = new int[5][8];
```

All rows have the same length

```
int [][] initWithElements = new int[][] {{2,4},{4,4,5,6,12}};
```

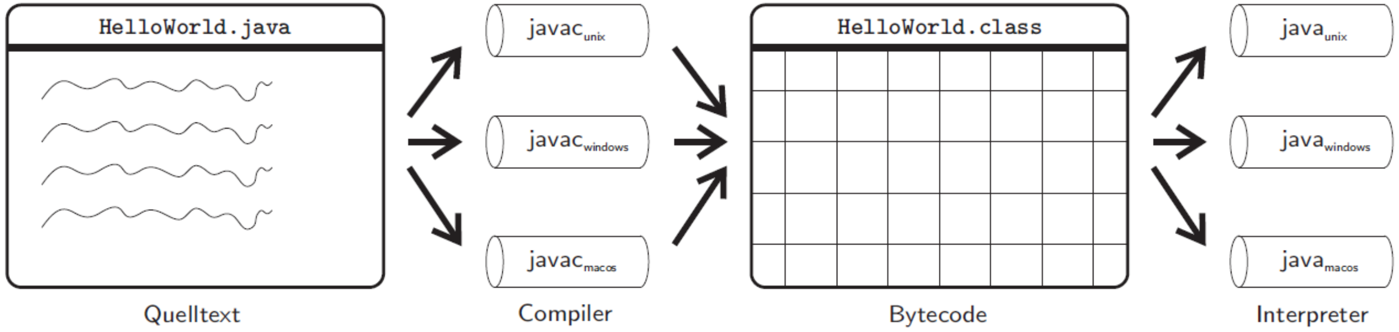
Learning Goals

- Getting to know differences between Heap and Stack
- Understanding relationship between objects and data types to main memory
- Understanding what happens when parameters are passed to a method
- Getting to know different methods to copy objects

Java-Virtual-Machine (JVM)
Stack and Heap

Java-Virtual-Machine (JVM)

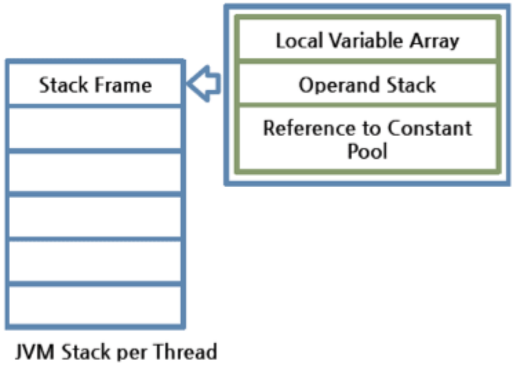
- Java is supposed to be independent of the platform



- JVM emulates a non-physical machine
  - Idea: Code is transformed for this non-physical machine (Java bytecode)
  - Emulation (interface between Java bytecode and physical machine) does depend on hardware

Java's Runtime Stack

- Part of main memory, where the JAVA stacks all method calls
- Roughly: All data that are not objects are stored in the stack
  - Local variables
  - Method parameters
  - Method calls

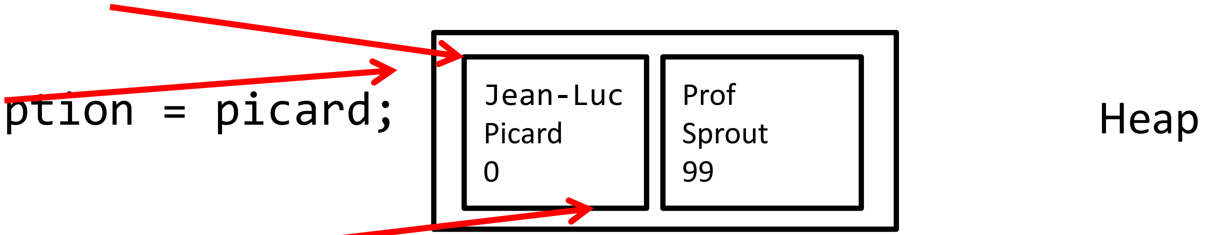


- With each method call, a new form is created on the stack
- When leaving the method, the form on top of the stack is removed

Heap

- Part of main memory that stores objects/instances of a class (i.e., everything that is created with **new**)

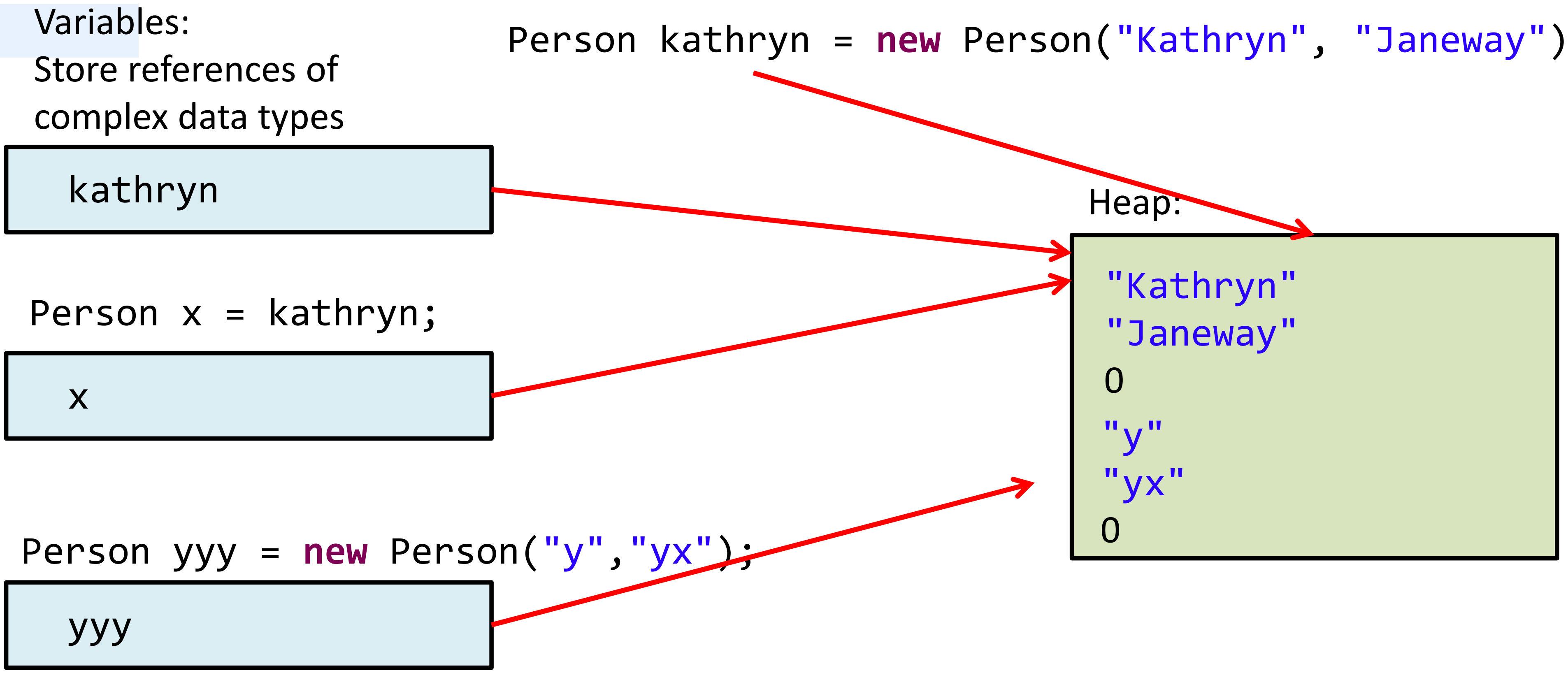
```
Person picard = new Person(„Jean-Luc“, „Picard“);

Person caption = picard;

Person sisko = new Person(„Benjamin“, „Sisko“,99);
```



- Memory is managed dynamically (reserving and freeing memory)
  - No memory leaks, so less work and error-prone
  - However: maybe performance issues

References

- Variable with class type holds a reference to an object



Variables:
Store references of complex data types

```
Person kathryn = new Person("Kathryn", "Janeway");
```

```
Person x = kathryn;
```

```
Person yyy = new Person("y","yx");
```

Recap: Comparison of Strings

- == returns **unexpected** result!
- Comparison with **`equals`**-method

```
String h = new String("Hi");
String t = new String("Hi");
if(h == t){
  System.out.println("Same!");
}
else {
  System.out.println("Not Same!");
}
```

Result: Not Same!

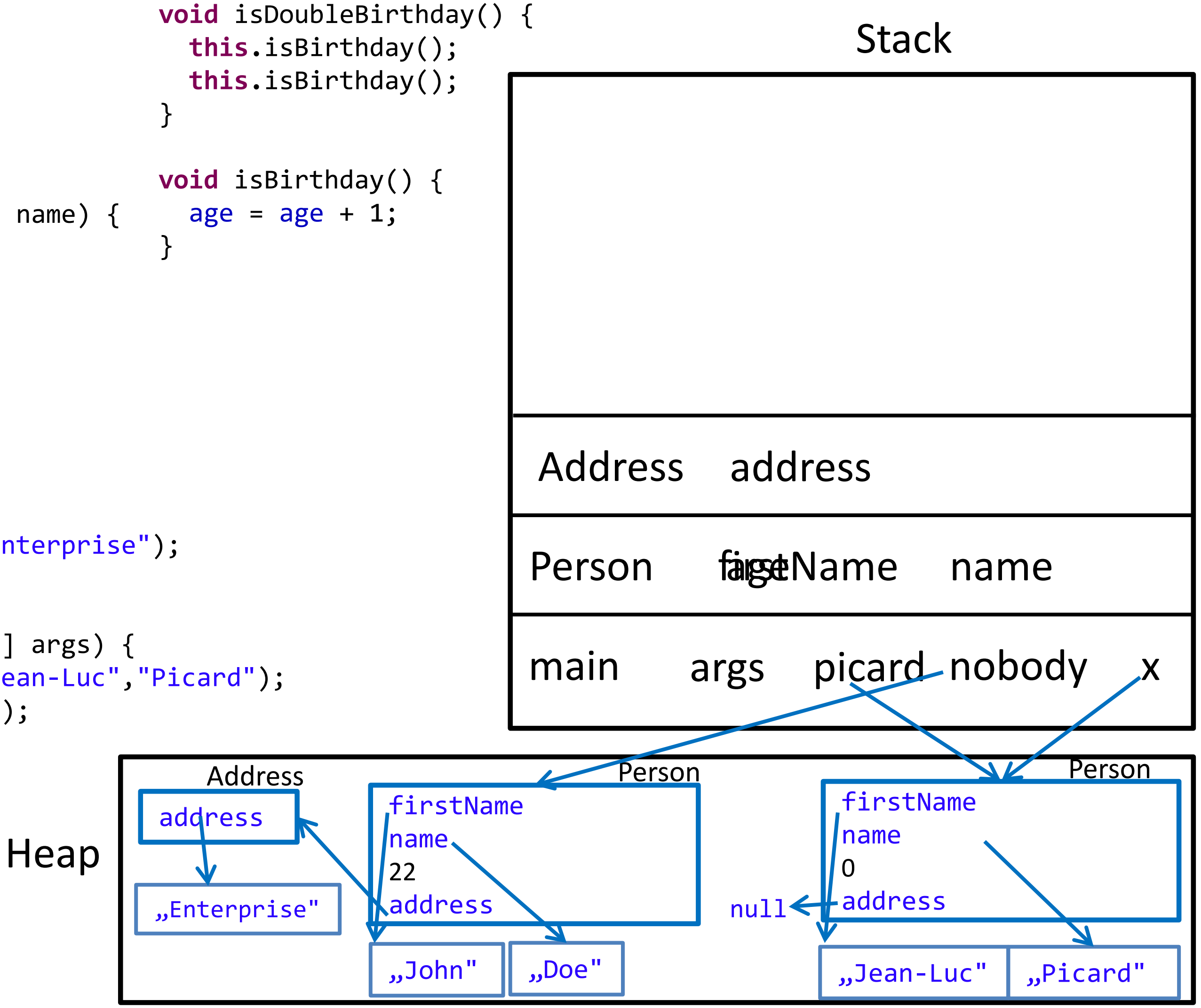
```
if(h.equals(t)){
  System.out.println("Same!");
}
else{
  System.out.println("Not Same!");
}
```

Result : Same!

Interaction of Heap and Stack

```
class Person {
  String firstName;
  String name;
  int age;
  Address address;

  Person(String firstName, String name) {
    this.firstName = firstName;
    this.name = name;
    this.age = 0;
  }

  Person(int age) {
    this.firstName = „John";
    this.name = „Doe";
    this.age = alter;
    this.address = new Address("Enterprise");
  }

  public static void main(String[] args) {
    Person picard = new Person(„Jean-Luc","Picard");
    Person nobody = new Person(22);
    Person x = picard;

    picard.isBirthday();
    x.isDoubleBirthday();
  }
}
```

```
void isDoubleBirthday() {
  this.isBirthday();
  this.isBirthday();
}

void isBirthday() {
  age = age + 1;
}
```



Garbage Collection

- Garbage Collector frees memory of objects that are not needed anymore
- Methods for cleaning before memory is freed
- `protected void finalize()`
  - Is counterpart of constructor
  - Is called automatically by Garbage Collector
  - Careful: Unclear when it is called (so avoid cleaning memory)
- No explicit freeing of objects, but via System.gc(), you can set a hint for the garbage collector

Quizz

```
public class Address {
  String name;
  boolean abroad = false;
  public Address(String name) {
    this.name = name;
  }
  public Address() {
    this.name = "Enterprise";
  }

  public void moveHouse(String newAddress, Person p) {
    if(name.equals(newAddress)) {
      System.out.println(„You stay on this ship.");
    }
    else {
      int index = registerNewAddress(newAddress);
      if (!abroad) {
        p.formerAddress[index] = this;
        this.name = neuerOrt;
      }
      else {
        p.formerAddress[index] = this;
        this.name = "N/A";
      }
    }
  }
  public int registerNewAddress(String address) {
    if(address.equals("Andromeda")) {
      abroad = true;
      return 2;
    }
    else {
       abroad = false;
       return 1;
}}}
```

```
class Person {
  String firstName;
  String name;
  int age;
  Address residence;
  Address[] formerResidences;
  Person (int age) {
    this.firstName = "Jane";
    this.name = "Doe";
    this.age = age;
    this.residence = new Address("Enterprise");
    formerResidences = new Address[2];
  }

  void movesHouse(String newResidence) {
   residence.moveHouse(newResidence, this);
  }

  public static void main(String[] args) {
    Person me = new Person(22);
    me.movesHouse("Andromeda");
  }
}
```

This program ends in an error. Where is the error? What is on the stack?

3 to 5 minutes

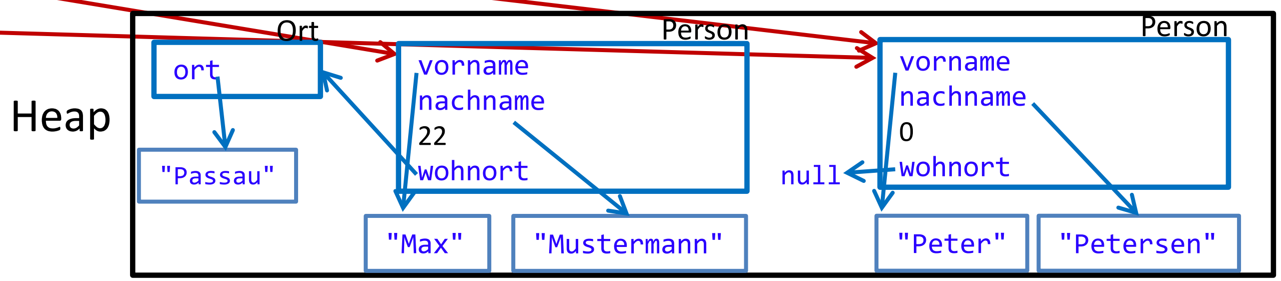
Input Parameters of Methods

References I

- There is an additional primitive data type – we did not explicitly look at it, but we used it

- Reference = „pointer“ to instances of classes
  - Variables that store complex types are references to according objects in memory

```
public static void main(String[] args) {
    Person peter = new Person("Peter","Petersen");
    Person nobody = new Person(22);
    Person x = peter;
}
```



References II

- Internally: a reference is an integer
  - Points to identy (address in memory) of object
  - Number corresponds to first memory cell that is used by the object
- Operator „==“ compares references, so only the addresses, not the state/content/value of object
- Arrays are also references!

Parameter Passing

- Call-by-Value
  - Values are copied in memory
  - Changes will be executed on copy
  - **No effects outside of a method!**

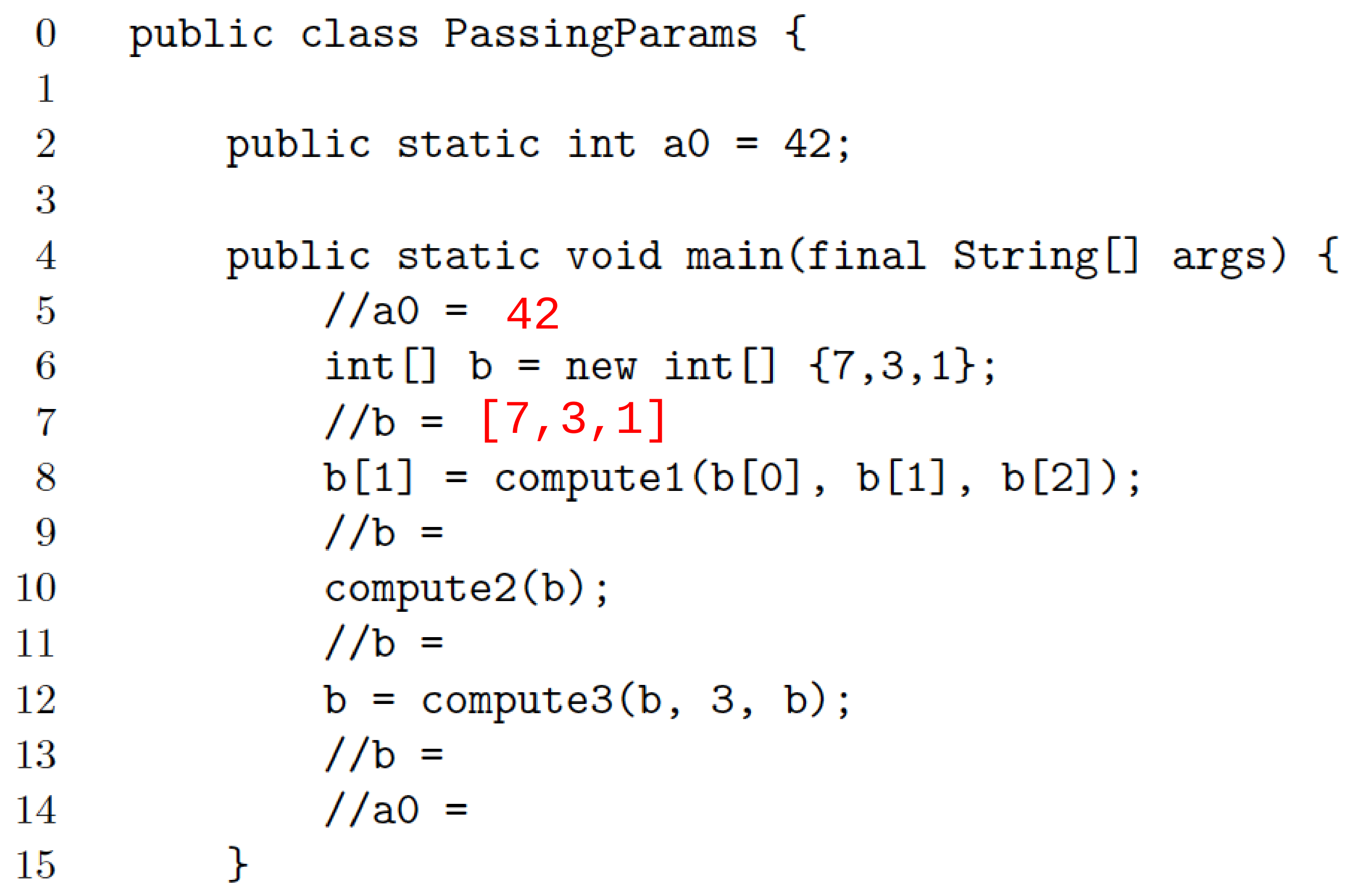
- Call-by-Reference
  - There is no copying
  - A pointer is only passed, and the pointer points to the place in memory that is changed within a method
  - **Changes of the values within method have an effect outside of the method**

Parameter Passing in Java

- Java always does call-by-value
  - Primitive data types are always copied (in the stack)
  - Changes do not have an effect outside of methods (because once a method is completed, everything is removed from the stack)

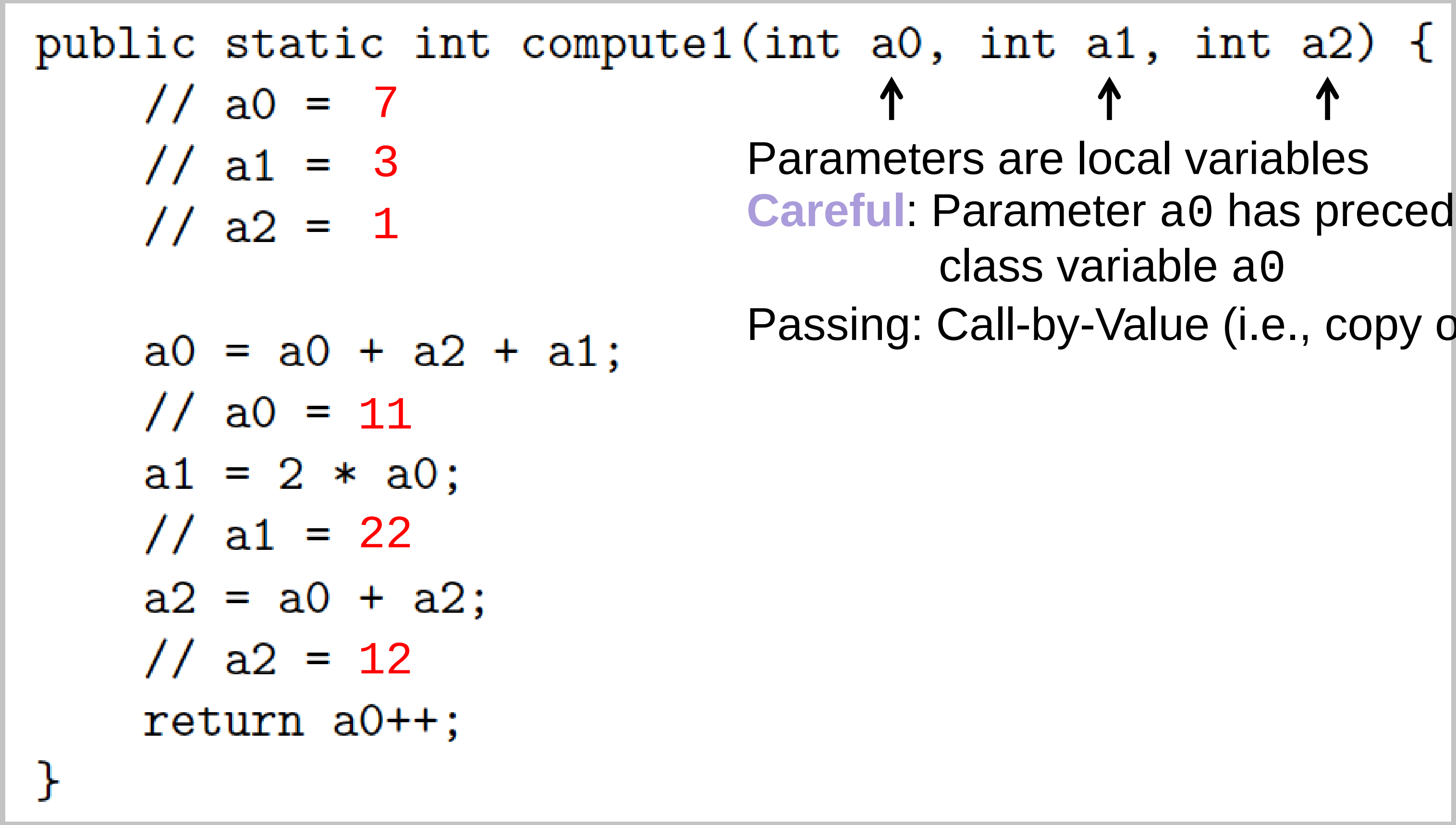
- Careful! If parameter is a reference, it is copied (in the stack)
  - Referenced object is not copied
  - May lead to unexpected side effects
  - Changes within method may have an effect outside of the method

Parameter Passing I



```
public class PassingParams {

    public static int a0 = 42;

    public static void main(final String[] args) {
        //a0 = 42
        int[] b = new int[] {7,3,1};
        //b = [7,3,1]
        b[1] = compute1(b[0], b[1], b[2]);
        //b =
        compute2(b);
        //b =
        b = compute3(b, 3, b);
        //b =
        //a0 =
    }
```

Parameter Passing II



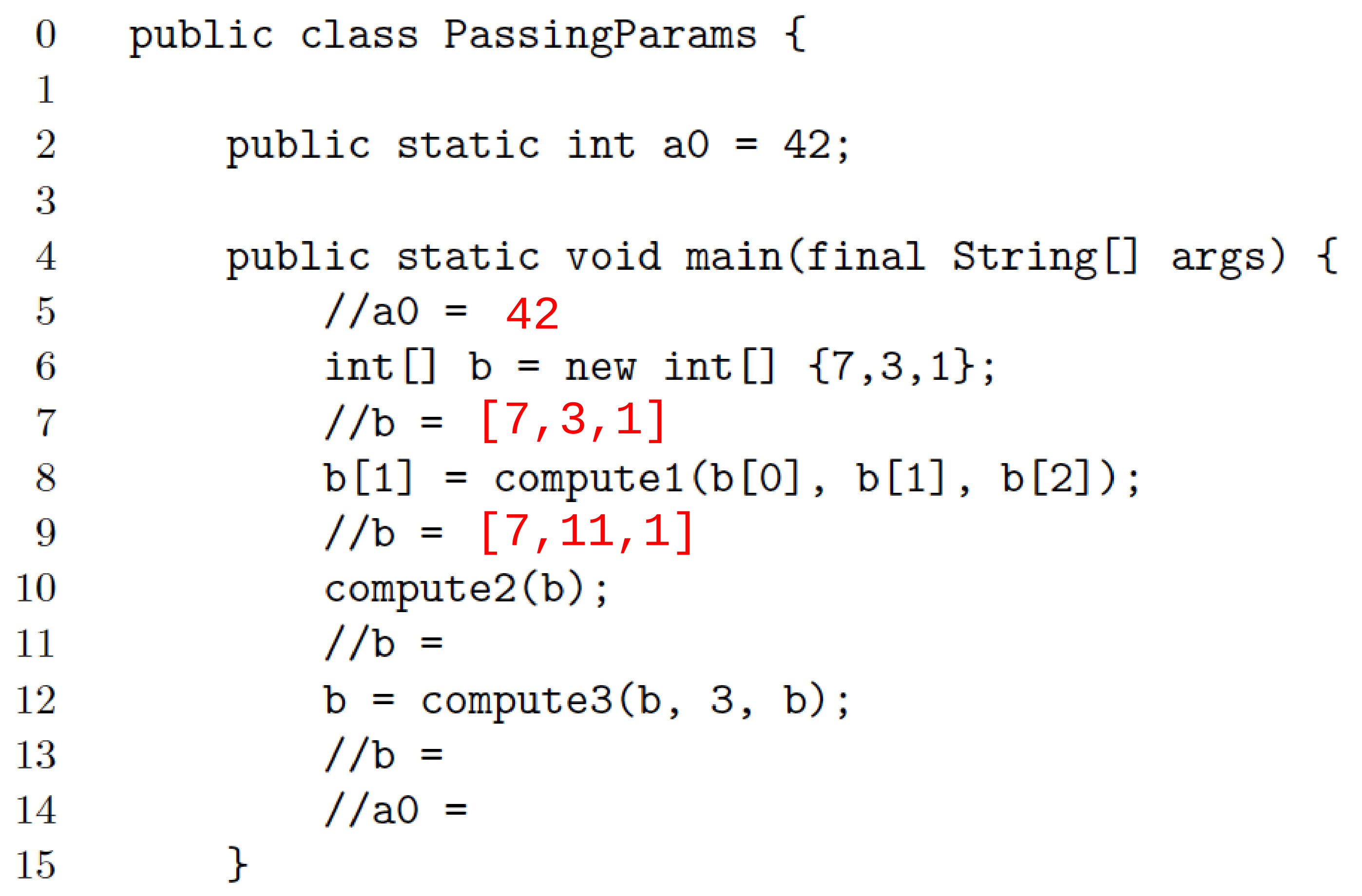
```
public static int compute1(int a0, int a1, int a2) {
    // a0 =  7
    // a1 =  3
    // a2 =  1

    a0 = a0 + a2 + a1;
    // a0 = 11
    a1 = 2 * a0;
    // a1 = 22
    a2 = a0 + a2;
    // a2 = 12
    return a0++;
}
```

Parameters are local variables

Careful: Parameter `a0` has precedence before class variable `a0`

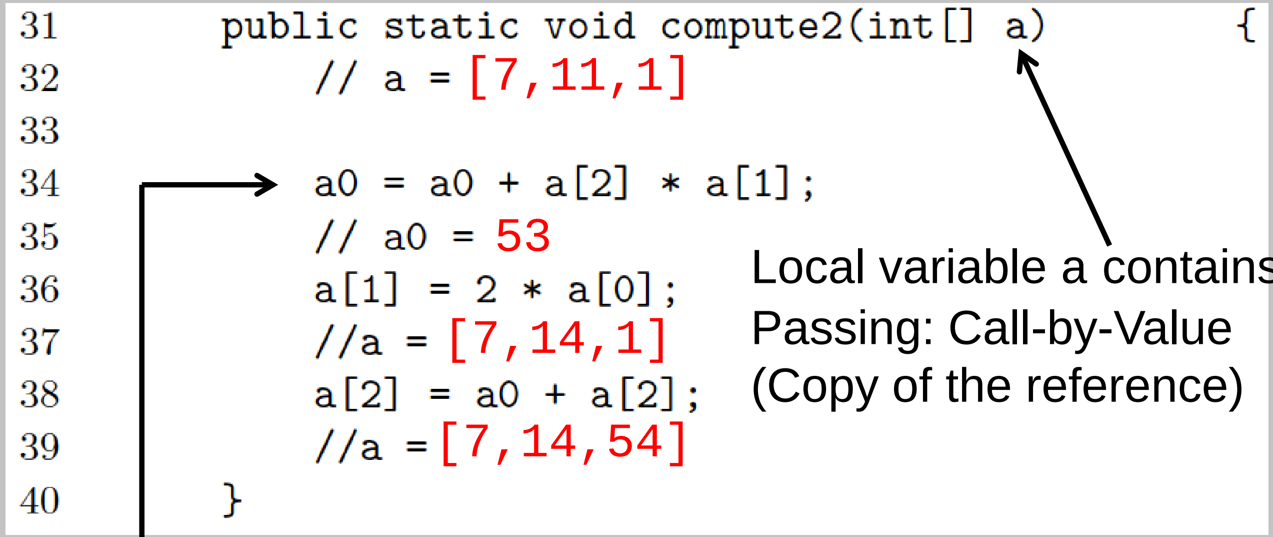
Passing: Call-by-Value (i.e., copy of the value)

Parameter Passing III



```
public class PassingParams {

    public static int a0 = 42;

    public static void main(final String[] args) {
        //a0 = 42
        int[] b = new int[] {7,3,1};
        //b = [7,3,1]
        b[1] = compute1(b[0], b[1], b[2]);
        //b = [7,11,1]
        compute2(b);
        //b =
        b = compute3(b, 3, b);
        //b =
        //a0 =
    }
```

Parameter Passing IV

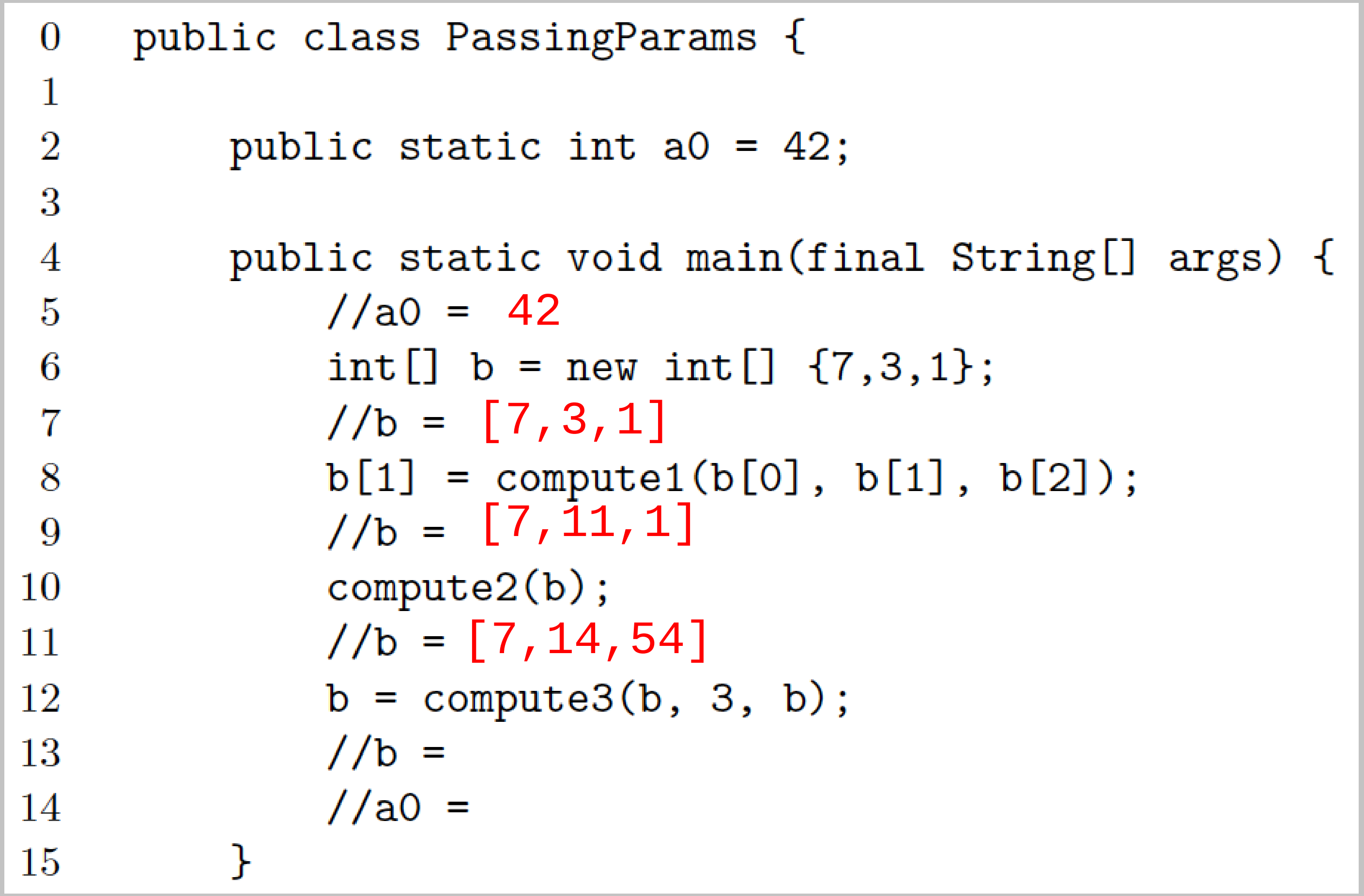


```
public static void compute2(int[] a)        {
    // a = [7,11,1]

    a0 = a0 + a[2] * a[1];
    // a0 = 53
    a[1] = 2 * a[0];
    //a = [7,14,1]
    a[2] = a0 + a[2];
    //a =[7,14,54]
}
```

Local variable a contains reference to b
Passing: Call-by-Value
(Copy of the reference)

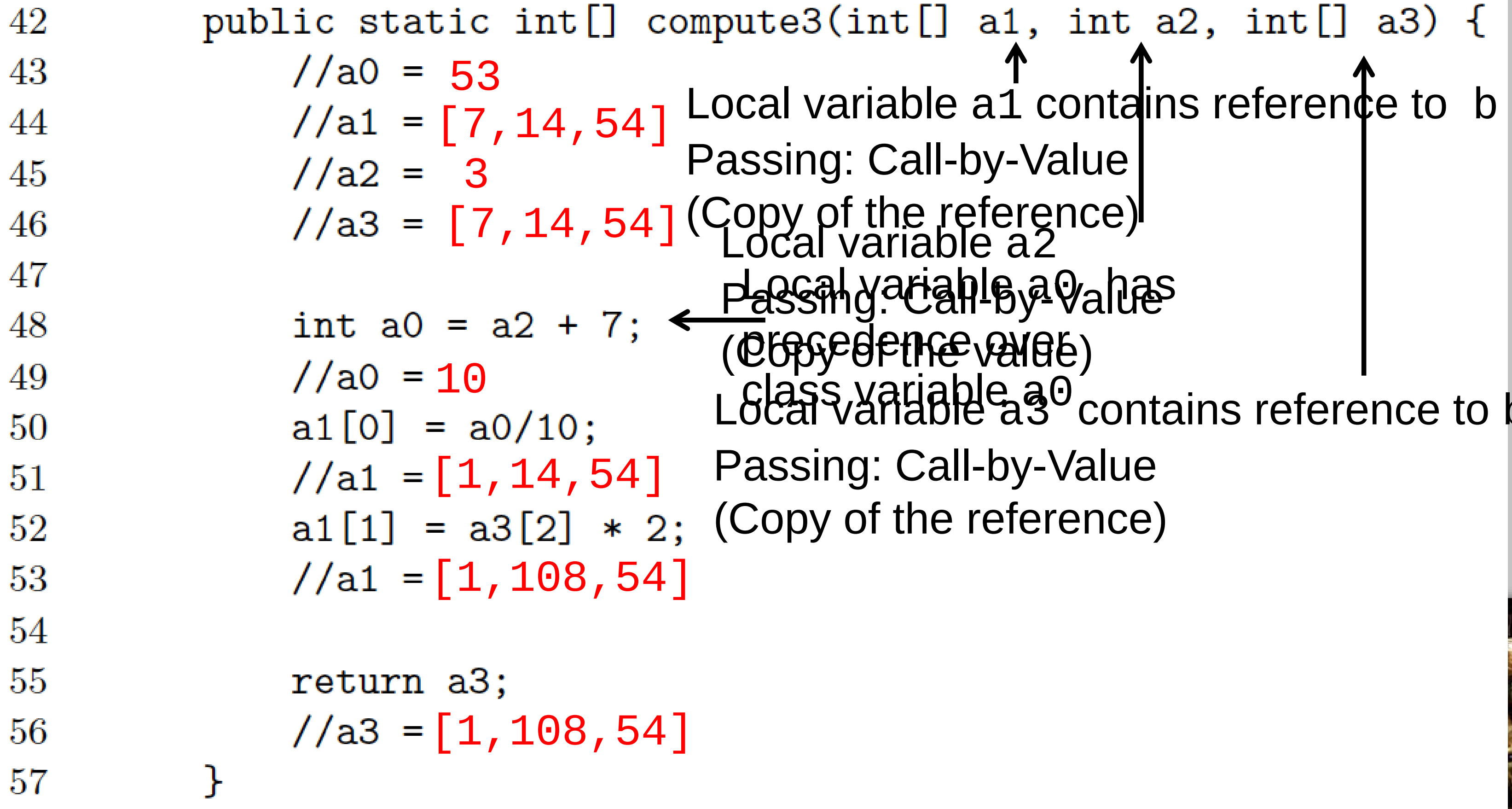
`a0` is class variable

Parameter Passing V



```
public class PassingParams {

    public static int a0 = 42;

    public static void main(final String[] args) {
        //a0 = 42
        int[] b = new int[] {7,3,1};
        //b = [7,3,1]
        b[1] = compute1(b[0], b[1], b[2]);
        //b = [7,11,1]
        compute2(b);
        //b = [7,14,54]
        b = compute3(b, 3, b);
        //b =
        //a0 =
    }
```

Parameter Passing VI



```
public static int[] compute3(int[] a1, int a2, int[] a3) {
    //a0 = 53
    //a1 = [7,14,54]
    //a2 =  3
    //a3 = [7,14,54]

    int a0 = a2 + 7;
    //a0 = 10
    a1[0] = a0/10;
    //a1 = [1,14,54]
    a1[1] = a3[2] * 2;
    //a1 = [1,108,54]

    return a3;
    //a3 = [1,108,54]
}
```

Local variable a1 contains reference to b

Passing: Call-by-Value

(Copy of the reference)

Local variable a2

Passing: Call-by-Value

(Copy of the value)

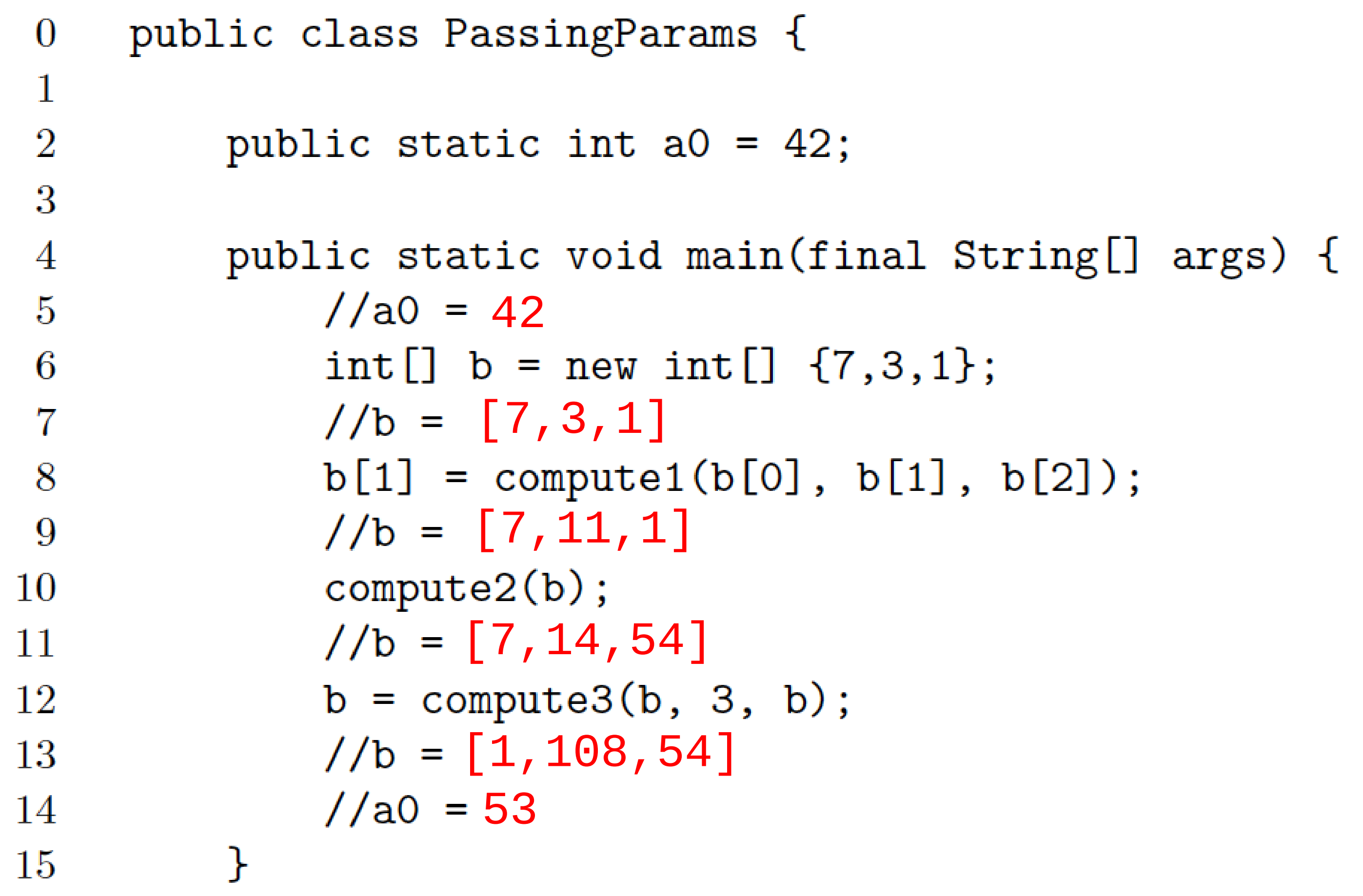
Local variable a0 has precedence over class variable a0

Local variable a3 contains reference to b

Passing: Call-by-Value

(Copy of the reference)

Parameter Passing VII



```
public class PassingParams {

    public static int a0 = 42;

    public static void main(final String[] args) {
        //a0 = 42
        int[] b = new int[] {7,3,1};
        //b = [7,3,1]
        b[1] = compute1(b[0], b[1], b[2]);
        //b = [7,11,1]
        compute2(b);
        //b = [7,14,54]
        b = compute3(b, 3, b);
        //b = [1,108,54]
        //a0 = 53
    }
```

Copying / Cloning Objects

Cloning of Objects

- Problem: We need more objects with exactly the same values/attributes
  - Example: A car has 1000 properties that have been defined in many methods (e.g., by customer or federal laws, et)
  - We need this car again, and setting everything again is too tedious or information is missing
  - What to do?
- Clone objects!
  - Idea: Get all values of an object to a new object
  - Two possibilities: shallow copy and deep copy

Shallow Copy

- Each class has the method `clone()`
- How it works:
  - New object is created in memory
  - All primitive types of object are copied
  - And this counts for references, as well!
- Effect: Pointer is copied and so attributes of both objects point to the same sub objects
  - May lead to unexpected side effects
  - Not recommended

Deep Copy

- Idea: implement own method that also clones all complex data types of object
- Advantage:
  - Changes to attributes have no effect on clones
  - Full control about what is to be cloned
- Disadvantage:
  - Implementation effort
  - Also the attributes need to provide a clone method

Implementation

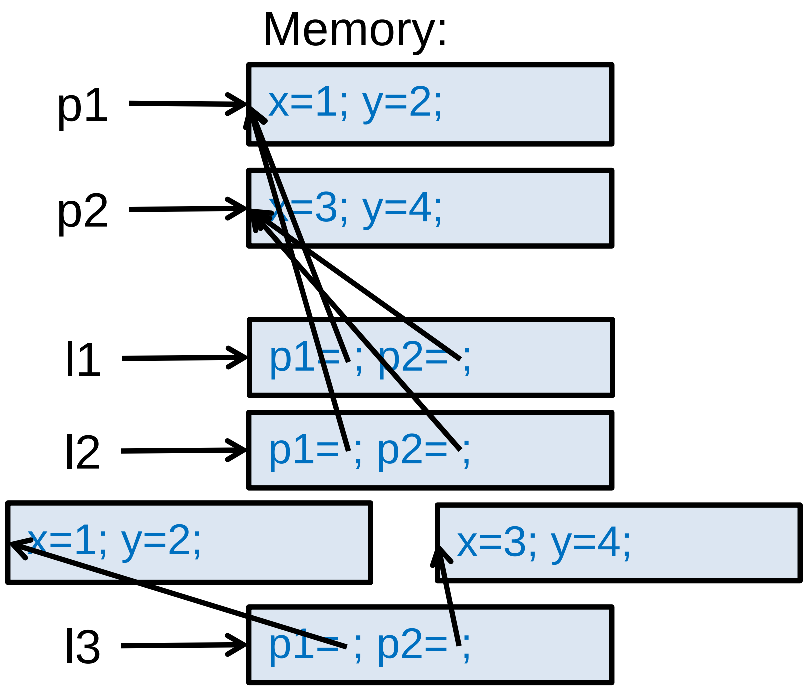
```
public class Point {
  int x,y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}

public class Line {
  Point p1,p2;
  public Line(Point p1, Point p2) {
    this.p1 = p1;
    this.p2 = p2;
  }
  public Line deepClone() {
    Point p1clone = new Point(p1.x, p1.y);
    Point p2clone = new Point(p2.x, p2.y);
    Line clone = new Line(p1clone, p2clone);
    return clone;
  }
}
```

```
public static void main(String[] args) {
  Point p1 = new Point(1,2);
  Point p2 = new Point(3,4);

  Line l1 = new Line(p1,p2);
  Line l2 = l1.clone();
  Line l3 = l1.deepClone();
}
```

„Behind the Scenes“

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

public class Line {
  Point p1,p2;
  public Line(Point p1, Point p2) {
    this.p1 = p1;
    this.p2 = p2;
  }
}

Point p1 = new Point(1,2);
Point p2 = new Point(3,4);

Line l1 = new Line(p1,p2);
Line l2 = l1.clone();
Line l3 = l1.deepClone();
```

l2 is new object in storage, but has no copied attributes. The attributes point to the same objects in memory as the original object.



Copy Constructor

- Alternative (best practice) known from C++
- Idea: Implement own constructor that receives an object of the same class

```
public class Complex {
  double real, imaginary;
  //Normal constructor
  public Complex (double re, double im) {
    this.real = re;
    this.imaginary = im;
  }

  //Copy constructor
  Complex (Complex c) {
    this.real = c.real;
    this.imaginary = c.imaginary;
  }
```

```
public static void main(String[] args) {
  //Instantiating object
  Complex c1 = new Complex (5, 5);

  //Copy constructor
  Complex c2 = new Complex (c1);

  //Now copy here. All non-primitive types
are only references
  Complex c3 = c2;
  }
}
```

New object in memory

Passed parameter has the same type as the class

Quiz!!!

What is on the heap, what is on the stack?

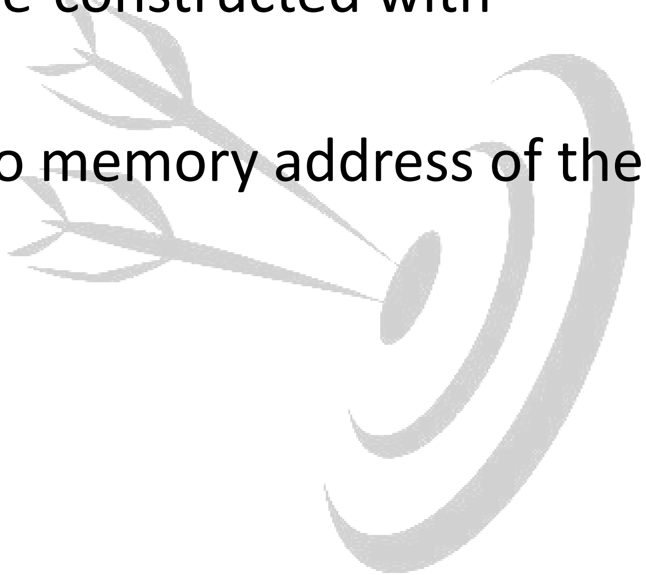
```java
public class Monster {
  private int numberTeeth 1 = 200; 2
  private String name; 3

  public static void main(String[] args) {
    Monster monster = 4 new Monster("Grarrar");        5
    monster.scrare();
```

```java
public class Monster {
  private int numberTeeth 1 = 200; 2
  private String name; 3

  public static void main(String[] args) {
    Monster monster = 4 new Monster("Grarrar");        5
    monster.scare();          6
    monster.chew();       7
    monster.flirt(new 8 Monster("Buuuuuhhh!"));      9
  }

  public Monster(String name) {
    this.name = name; 10
  }

  public void scare() {
    int soundVolume 11 = 5; 12
    String scream 13 = "AAHHHHHHHHAAAA!!!111";        14
    System.out.print(this.name + scream);
    for (int i 15 = 0; i < soundVolume; i++) {
        System.out.print("!");
    }
    System.out.println("");
  }

public void chew() {
  for(int i = 0; i < this.numberTeeth / 4; i++) {
    System.out.print("Grind"); 16
  }
  System.out.println("");
}

public void flirt(Monster monster) {
  this.scare();
  monster.scare();
}
}
```

3 to 5 minutes

Take Aways I

- There are two memory in the Java Virtual Machine
  - Heap stores all objects (i.e., instances of complex data types)
  - Stack stores primitive types, methods calls, etc.
- The stack is constructed with every method call und de-constructed with completed methods
- Variables of complex types are references (pointers) to memory address of the object in the heap



Take Aways II

- There are two ways of parameters passing
  - Call by values passes the value
  - Call by reference passes the pointer to the value
- Java only uses call by value, but with references, the pointer is copied
- Objects can be copied via cloning
  - Shallow Copy copies only "top level" of an object
  - Deep copy copies attributes of complex types