# Objects and Classes

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.jku.at
http://www.risc.jku.at

# Contents

# Motivation

How can a function return multiple results?

- Function may have multiple parameters but only one result.
  - Cannot write a function that returns the day, month, year of a date.
  - Idea: pack day, month, year into a single object of type `Date`.
  - `Date` is an object type, i.e., a class.
- Classes are used in two ways:
  1. As a module, i.e., a collection of methods (that's what we did up to now).
  2. As the type of an object, i.e., a collection of values.
- We will see later why it makes sense to use classes in two ways.

We will now investigate classes as object types.

# Class Declaration

How is an object type declared?

```
public class Date
{
  public int day;
  public String month;
  public int year;
}
```

- Collection of variables.
  - We call the variables object fields or attributes.

An object type is a collection of fields.

# Object Creation

How is an object created?

```
Date date = new Date();
```

- Object variable declaration and initialization:
  - Variable declaration: `Date date`.
  - Object creation: `new Date()`.

An object is created as an instance of a class.

# Object Use

How are the object fields used?

```
date.day = 24;
date.month = "December";
date.year = 2001;

System.out.println(date.day);
```

- Object fields can be read and written like normal variables.
  - Qualification by object: `date.field`.

Fields can be referenced by qualification with the object.

# Multiple Function Results

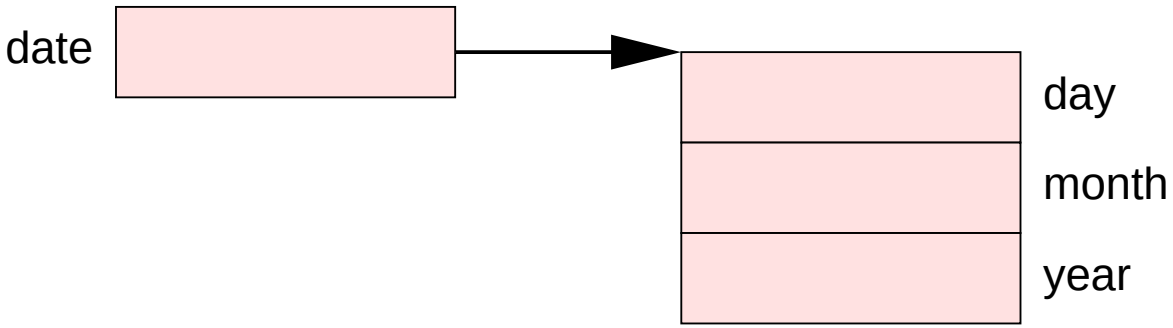
```
public static void main(String[] args)
{
  Date d = readDate();
  System.out.println("date: " + d.day + ". " + d.month + " " + d.year);
}

public static Date readDate()
{
  Date date = new Date();
  date.day = Input.readInt();
  date.month = Input.readString();
  date.year = Input.readInt();
  return date;
}
```

Let us now look at the details of objects and their use.

# Objects

How is an object represented in computer memory?

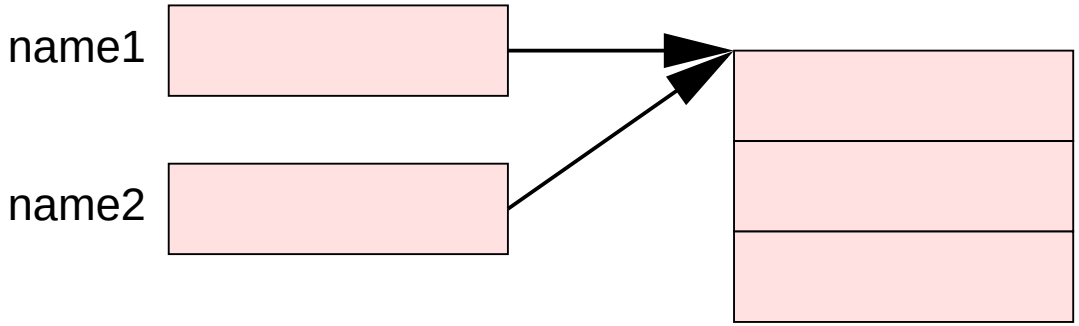



- An object is represented as a pointer.
  - Address of the memory area that holds the object data.
  - Java: object pointers are 32 bit addresses.
- A Java variable of an object type references the object data.
  - Variable holds a pointer to the object.
  - Variable does not hold the object data.

# Object Assignment

What happens if we assign an object to another variable?

```
Class name1 = new Class();
Class name2 = name1;
```




An object may be referenced by multiple names.

# Example

Program:

```
Date date1 = new Date();
date1.day = 21;
Date date2 = date1;
System.out.println(date2.day);
date1.day = 17;
System.out.println(date2.day);
```

Output:

```
21
17
```

The modification of an object is visible to all object variables.

# Object Comparison

What happens if we compare two objects?

```
Date d1 = new Date();
d1.day = 24; d1.month = "December"; d1.year = 2001;
Date d2 = new Date();
d2.day = 24; d2.month = "December"; d2.year = 2001;
System.out.println(d1 == d2);
d2 = d1;
System.out.println(d1 == d2);
```

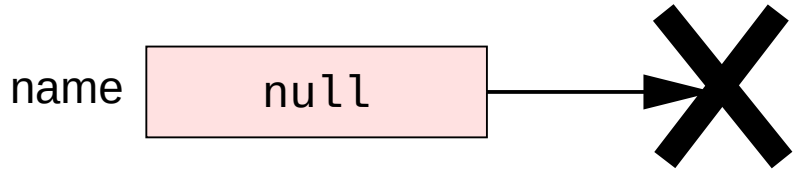
```
false
true
```

The object pointers are compared, not the object data.

## The `null` Reference

The reference `null` does not refer to an object.

*name* `= null;`




- `null` is the default value for any object type.
  - `null` is a special value that is compatible with every object type.
  - This value is used if the initialization part is missing in a variable declaration.

Any access to an object field via `null` yields a runtime error.

## Example

Program:

```
Date date = null;
System.out.println(date.year);
```

Runtime Error:

```
Exception in thread "main" java.lang.NullPointerException
        at Main.main(Main.java:6)
```

If an object reference can be null, you have to test it before you may access the object fields.

## Classes

A public class defines the memory layout for all objects of this class.

```
public class Name
{
  public type name = expression;
  public final type name = expression;
  ...
}
```

- Variable and constant declarations.
  - Initialization expression defines initial field value on object creation.
  - In variable declarations, initialization may be omitted; then the default value is used.

Any instance of this public class has all the declared object fields.

## Class Declarations

Where is the class declaration written?

- In a separate file *Class*.java.
  - Automatically compiled to *Class*.class.
- Into the source code of a program class (module).

```
public class Module
{
  public static class Class
  {
    ...
  }
  ...
}
```

  - *Class* can be used by other modules only as *Module.Class*.

## Qualifed Names

Object fields can be referenced by qualified names.

- Read (constant or variable) field value.

```
... = ... object.name ...;
```

- Write (variable) field value.

```
object.name = ...
```

Use like any other variable respectively constant.

# Object Creation

An object is created by a `new` expression.

```
new Class()
```

- A `new` expression
  1. allocates the memory for an object of the denoted class,
  2. initializes all object fields with the values specified in the class (respectively with default values, if there were no values specified), and
  3. returns a pointer to this object.
- Should be only used in declarations or assignments.

```
Class name = new Class();
name = new Class();
```

Do not use object creation expressions within other expressions.

## Constructor Calls

How can we initialize a new object?

- It is rather inconvenient to initialize an object by assignments.

```
Date date = new Date();
date.day = 24;
date.month = "December";
date.year = 2001;
```

- The same effect may be achieved by a constructor call.

```
Date date = new Date(24, "December", 2001);
```

We need to declare a corresponding constructor.

# Constructor Declaration

```
public class Date
{
  public int day;
  public String month;
  public int year;
  public Date(int d, String m, int y)
  {
    day = d;
    month = m;
    year = y;
  }
}
```

- Constructor is a special method in a class.
  - A constructor has the same name as the enclosing class.
  - A constructor is executed when an object is created by a corresponding `new` call.

## Constructor Behavior

A constructor is different from a normal program method.

- A constructor is bound to a particular object of the class.
  - The constructor is bound to the object created by the `new` call.
  - The constructor may read and write the fields of this object without qualification.
- The system provides a default constructor.
  - Constructor *`Class`*`()` which initializes variables to default values.
- User-defined constructor is executed after the default constructor.
  - Fields are already initialized with default values when constructor body is executed.
- If constructor is defined, default constructor can't be used in `new`.
  - User must explicitly provide constructor *`Class`*`()` (whose body may be empty).

A constructor is bound to an object, not to a class.

# Initialization of Constant Fields

```
public class Date
{
  public int day;
  public String month;
  public final int year;

  public Date(int d, String m, int y)
  {
    day = d;
    month = m;
    year = y;
  }
}
```

- Constructor may assign value to (unitialized) constant field.
  - After assignment, constant cannot be changed any more.

## The `this` Pointer

Within a constructor, `this` refers to the current object.

```
public class Date
{
  public int day;
  public String month;
  public int year;

  public Date(int day, String month, int year)
  {
    this.day = day;
    this.month = month;
    this.year = year;
  }
}
```

## Constructor Overloading

There may be multiple constructors with different parameter lists.

```
public class Date
{
  ...
  public final int THIS_YEAR = 2001;
  public Date(int day, String month)
  {
    this.day = day;
    this.month = month;
    this.year = THIS_YEAR;
  }
}

Date date1 = new Date(31, 12);
Date date2 = new Date(31, 12, 1999);
```

# Calling Constructors within Constructors

- First statement of constructor body may call another constructor.

  `this(`*`expressions`*`);`

  – *Class(expressions)* is called rather than default constructor before remaing body is executed.

```
public class Date
{
  ...
  public final int THIS_YEAR = 2001;

  public Date(int day, String month)
  {
     this(day, month, THIS_YEAR);
  }
}
```

## Transient Parameters

Java does not provide transient parameters.

- The value of a transient parameter is changed by the method.
  - Argument has after the call a different value than before the call.

```
// increase the value of i by n
public static void increment(int i, int n);
```

- This cannot be achieved by simple parameter assignment.
  - Why does this not work?

```
public static void increment(int i, int n)
{ i = i+n; }
```

Objects can be used to simulate transient parameters.

# Transient Parameters

Encapsulate transient parameter into an object.

```
public class Integer
{
  public int value;
  public Integer(int value)
  { this.value = value; }
}

public static void increment(Integer i, int n)
{ i.value = i.value + n; }

Integer i = new Integer(5);
increment(i, 3);
System.out.println(i.value);
```

## Structuring Programs

- Programs are not written as monolithic pieces of code.
  - Not just a single method `main()`.
- Programs are composed from multiple methods.
  - Each method provides part of the functionality.
  - All methods collaborate to achieve overall goal.
- Methods correspond to steps in gradual refinement process.
  - Do not insert the derived program parts into the main program.
  - Make each program part a separate method.

Methods and objects are used to give structure to programs.

# Analyzing Exam Grades: Overall Structure

```
public static void analyzeExamGrades()
{
  char answer = 0;
  do
  {
    Result result = analyzeExam();
    printResult(result);
    answer = askToContinue();
  }
  while (answer == 'y');
}
```

Introduce variable to hold result of analysis.

# Analyzing Exam Grades: Asking the User

```
public static char askToContinue()
{ while(true)
  {
    System.out.print("Another exam (y/n)? ");
    char answer = Input.readChar();
    if (Input.isOkay() && (answer == 'y' || answer == 'n'))
      return answer;
    if (Input.hasEnded())
    {
      System.out.println("Unexpected end of input!");
      System.exit(-1);
    }
    System.out.println("Invalid input!");
    Input.clearError();
  }
}
```

# Analyzing Exam Grades: Analyzing an Exam

```
public static Result analyzeExam()
{
  Result result = new Result();

  // read and process exam
  while (true)
  {
    int grade = readGrade(result.count);
    if (grade == 0) return result;
    processGrade(grade, result);
  }
}
```

# Analyzing Exam Grades: Processing a Grade

```
public class Result
{
  public int count = 0;                // number of grades
  public int sum = 0;                  // sum of grades processed so far
  public int min = Integer.MAX_VALUE; // minimum grade processed so far
  public int max = Integer.MIN_VALUE; // maximum grade processed so far
}

public static void processGrade(int grade, Result result)
{
  result.count++;
  result.sum += grade;
  if (grade < result.min) result.min = grade;
  if (grade > result.max) result.max = grade;
}
```

# Analyzing Exam Grades: Reading a Grade

```
public static int readGrade(int count)
{ while(true)
  {
    System.out.print("Enter grade "+(count+1)+" (0 when done): ");
    int grade = Input.readInt();
    if (Input.isOkay() && 0 <= grade && grade <= 5)
      return grade;
    if (Input.hasEnded())
    {
      System.out.println("Unexpected end of input!");
      System.exit(-1);
    }
    System.out.println("Invalid input!");
    Input.clearError();
  }
}
```

# Analyzing Exam Grades: Printing the Result

```
public static void printResult(Result result)
{
  System.out.println("\nNumber of grades: " + result.count);

  // these values are only valid, if there was at least one grade
  if (result.count > 0)
  {
    System.out.println("Best grade: " + result.min);
    System.out.println("Worst grade: " + result.max);
    System.out.println("Average grade: " +
                       (float)result.sum/(float)result.count);
  }

  System.out.println("");
}
```

# Analyzing Exam Grades: Complete Program

```
public class Exams2
{
  public static void main(String[] args)
  { analyzeExamGrades(); }

  public static public class Result { ... }

  public static void analyzeExamGrades() { ... }
  public static Result analyzeExam() { ... }
  public static int readGrade(int count) { ... }
  public static void processGrade(int grade, Result result) { ... }
  public static void printResult(Result result) { ... }
  public static char askToContinue() { ... }
}
```