Objects with Methods

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

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

Classes as Modules

A class may be a collection of methods, variables, and constants.

```
public class Class
{
  public static type variable = expression;
  public static final type constant = expression;
  public static type method(parameters)
  { ... }
  ...
}
```

- A method is bound to a class:

```
Class.method(arguments)
```

- Method may refer to variables/constants also bound to the class.

Classes as Object Types

A class may be the type of an object.

```
public class Class
{
  public type variable = expression;
  public final type constant = expression;
  public Class(parameters)
  { ... }
  ...
}
```

- Call of constructor bound to an object:

```
new Class(arguments)
```

- Constructor can refer to variables/constants (fields) of the object.

Class Declarations

Difference is the use of the keyword `static`.

- `static` declarations.
  - Declared entity is bound to the class in which it is declared.
  - Methods, global constants/variables.
- non-`static` declarations.
  - Declared entity belongs to an object of this class.
  - Constructors, object fields.

Any entity may be declared as `static` or as non-`static`.

Object Methods

A method may be bound to an object.

```
public class Class
{
   public type method(parameters)
   {
      ...
   }
   ...
}
```

- Declaration of an object method.
  - May refer to the fields of the object to which it is bound.
- Invocation by a qualified name:

```
object.method(arguments)
```

Example

```
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;
  }

  public int getYear() { return year; }
  public void setYear(int year) { this.year = year; }
  public void increaseYear() { year++; }
}
```

Example

```
Date date = new Date(24, "December", 2001);
System.out.println(date.getYear());
```

```
2001
```

```
date.setYear(date.getYear()+1);
System.out.println(date.getYear());
```

```
2002
```

```
date.increaseYear();
System.out.println(date.getYear());
```

```
2003
```

Object Fields versus Object Methods

Why not refer to the object fields directly?

- Two possibilities:
  - `date.year = date.year+1;`
  - `date.increaseYear();`
- By use of methods, object type becomes an abstract datatype.
  - Public interface: object methods.
  - Private implementation: object fields.
  - Implementation can be changed at any time without affecting the interface.
  - Object can be used without knowing the implementation.

Only refer to object fields from methods within the object.

Analyzing Exam Grades: Abstracting Data

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

  public int getCount() { return count; }
  public void process(int grade)
  { count++;
    sum += grade;
    if (grade < min) min = grade;
    if (grade > max) max = grade;
  }
  void print() { ... }
}
```

Analyzing Exam Grades: Abstracting Data

```
public void print()
{
  System.out.println("\nNumber of grades: " + count);

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

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

Analyzing Exam Grades: General Structure

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

How a result is printed is hidden within `Result`.

Analyzing Exam Grades: Analyzing an Exam

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

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

How a result is processed is hidden within `Result`.

General Class Structure

```
public class Class
{
  public static type variable = expression;        // class fields
  public static final type constant = expression;

  public static type method(parameters) { ... }    // class methods

  public type variable = expression;               // object fields
  public final type constant = expression;

  public Class(parameters) { ... }                 // constructors

  public type method(parameters) { ... }           // object methods
}
```

The `this` Pointer

Which entity can refer to (read/write/call) which other entity?

- Special Java constant:

  `this`

- `this` refers to the "current object".
  - Can be used in object field declarations, constructors, and object methods.

Answer is based on the `this` pointer.

Static versus Non-static Entities

Short explanation (for details, see the lecture notes).

1. Object fields/methods can be only used by object qualification.
   - `object.method()`
2. For unqualified access, we need the `this` pointer.
   - Automatic translation: `method()` → `this.method()`.
3. Object field declarations and object methods receive `this`.
   - They can therefore refer to object fields and call object methods in the "current" object.
4. Class field declarations and class methods do not receive `this`.
   - There is no "current" object to which they can refer.

Entities declared as `static` cannot use `this`.

Non-static Entities

Where does the `this` pointer come from?

- Constructors: from the JVM.
  - When an object has been allocated, its pointer is passed to the constructor as `this`.
- Object field declarations: from the default constructor.
  - All initializations are performed by the default constructor.
  - The default constructor receives the `this` pointer from the JVM.
- Object methods: from the caller.
  - Object call `object.method()` → `this = object`.
  - The implicit parameter `this` in `method` is set to `object`.
  - When calling another method of the same object, the pointer is implicitly forwarded.

    `method()` → `this.method()`.

Example: Object Fields

```
public class Numbers
{
  public int zero = 0;
  public int one = zero+1;
}
```

→

```
public class Numbers
{
  public int zero = 0;
  public int one = this.zero+1;
}
```

Legal declaration.

Example: Class Fields

```
public class Numbers
{
  public int zero = 0;
  public static int one = zero+1;
}

Numbers.java:4: non-static variable zero cannot be referenced
                from a static context
    static int one = zero+1;
                     ^
```

one does not have this.

Example: Class Fields

```
public class Numbers
{
  public static int zero = 0;
  public int one = zero+1;
}
```

$\rightarrow$

```
public class Numbers
{
  public static int zero = 0;
  public int one = Numbers.zero+1;
}
```

Legal declaration.

Example: Object Methods

```
public class Numbers
{
  public int one = 1;
  public int add1(int n) { return n+one; }
}
```

→

```
public class Numbers
{
  public int one = 1;
  public int add1(int n) { return n+this.one; }
}
```

Legal declaration.

Example: Object Methods

```
public class Numbers
{ public int mult(int n, int m)
  {
    return n*m;
  }
  public int square(int n)
  {
     return mult(n, n);
  }
}
```

→

```
public int square(int n) { return this.mult(n, n); }
```

Legal declaration.

Example: Class Methods

```
public class Numbers
{
  public int one = 1;
  public static int add1(int n)
  {
    return n+one;
  }
}

Numbers.java:6: non-static variable one cannot be referenced
                from a static context
    return n+one;
              ^
```

add1 does not have `this`.

Example: Class Fields

```
public class Numbers
{
  public int one = add(0, 1);
  public static int add(int n, int m) { return n+m; }
}
```

→

```
public class Numbers
{
  public int one = Numbers.add(0, 1);
  public static int add(int n, int m) { return n+m; }
}
```

Legal declaration.