Concepts of Object-Oriented Programming

Prof. Dr. Peter Müller

Chair of Programming Methodology

Exercises 3: Java and other OO Languages

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Agenda for Today

- Finish Java Overview
- Quick look at Scala, Squeak, C++, and C#

Core Concepts: Summary

- Core concepts of the OO-paradigm
  - Object model
  - Interfaces and encapsulation
  - Classification and polymorphism
- Core concepts are **abstract concepts** to meet the new requirements
- To apply the core concepts we need ways to **express them in programs**
- **Language concepts** enable and facilitate the application of the core concepts

Subtyping

- **Substitution principle**
  Objects of subtypes can be used wherever objects of supertypes are expected

- **Subtype polymorphism**
  Program parts working with supertype objects work as well with subtype objects

Inheritance versus Subtyping

- **Subtyping** expresses **classification**
- **Inheritance** is a means of **code reuse**
- Inheritance is **usually coupled** with subtyping
  - Terminology: **Subclassing** = Subtyping + Inheritance

- Issues
  - Subtyping without inheritance
  - Inheritance without subtyping

Rules for Subtyping: Summary

- Subtype objects must **fulfill contracts** of supertypes, but
  - Subtypes can have **stronger invariants**
  - Overriding methods of subtypes can have
    weaker preconditions
    stronger postconditions
    than corresponding supertype methods

- Concept is called **Behavioral Subtyping**
- Consequence of substitution principle

CAT calls in Eiffel

- Changed Availability or Type
- Removing a feature from the interface
- Using a subtype for a parameter in an overriding feature
- Behavior not specified
- Depends on the Eiffel compiler used

```
class SUPER

feature
  f is
  do
    print("Hello World!%N")
  end

  g(p: SUPER) is
  do
    print("g in SUPER%N")
  end
end -- class SUPER
```

class SUB

```
inherit SUPER undefine f redefine g end
feature {NONE}
  f is do
    print("My private message%N")
  end

feature
  g( p: SUB ) is do
    print("g in SUB%N")
    p.subm
  end
  subm is do
    print("New feature of SUB called!%N")
  end
```

Using the classes

```
local
   sup:  SUPER
   sup2: SUPER
   sub:  SUB
do
   create sub
   sup := sub
   sup.f
   sup.g( sup )
   create sup2
   sup.g( sup2 )
end
```

Exercise 3.1 – Behavioral Subtyping

- Java is invariant in the parameter types and type arguments
- However, the behavior of a subtype could still violate behavioral subtyping

```
class Super {
  void m(Object o) {/* always ok */ }
}
class Sub extends Super {
  void m(Object o) {
    if (!o instanceof MyType)
      thrown new IllegalArgumentException();
    ... }}
```

Exercise 3.2

- Improved code:

```
@Override
public boolean equals(Object obj) {
  if (obj instanceof StaticCallVariable) {
    StaticCallVariable scv =
       (StaticCallVariable) obj;
    return callingMethod.equals(scv.callingMethod)
      && calledMethod.equals(scv.calledMethod)
      && instrLocation == scv.instrLocation;
  }
  return false;
}
```

Exercise 3.3: Overriding Example – Setup

```
class Upper {}
class Middle extends Upper {}
class Lower extends Middle {}

class Super {
  void foo( Middle a1 ) {
    System.out.println(
   "Super.foo("+a1+")");
  }
}
class Sub extends Super {
  void foo( Upper a1 ) {
    System.out.println(
   "Sub.foo("+a1+")");
  }
}
```

```
Super super1;
Sub   sub1;
Lower lower1 =
    new Lower();
Upper upper1 =
    new Upper();
```

Overriding Example – Main

```
Super: foo(Middle a1)
Sub:   foo(Upper a1)
```

```
System.out.println("Calls on Super object:");
      super1 = new Super();
  1   super1.foo( lower1 );
  2   super1.foo( upper1 );

System.out.println("\nCalls on Sub object in
  Super reference:");
      super1 = new Sub();
  3   super1.foo( lower1 );
  4   super1.foo( upper1 );

System.out.println("\nCalls on Sub object in
  Sub reference:");
      sub1 = new Sub();
  5   sub1.foo( lower1 );
  6   sub1.foo( upper1 );
```

Compilation Error!

Compilation Error!

Overriding Example - Output

```
Super: foo(Middle a1)
Sub:   foo(Upper a1)
```

```
Calls on Super object:
1 Super.foo(Lower@765291)

Calls on Sub object in Super reference:
3 Super.foo(Lower@765291)

Calls on Sub object in Sub reference:
5 Super.foo(Lower@765291)
6 Sub.foo(Upper@26e431)
```

Scala

- There are many more programming languages that compile down to Java Bytecode
- One “hot” language is Scala

  `http://www.scala-lang.org/`

- “Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of object-oriented and functional languages. It is also fully interoperable with Java.”

Scala – Type Inference

```
case class MyPair[A, B](x: A, y: B);


object InferenceTest3 extends
  Application {
   def id[T](x: T) = x
   val p = new MyPair(1, "scala")
   val q = id(1)
}
```

Scala – Pattern Matching

```
abstract class Tree
case class Branch(left: Tree,
                  right: Tree) extends Tree
case class Leaf(x: Int) extends Tree

def sumLeaves(t: Tree): Int = t match {
  case Branch(l, r) =>
       sumLeaves(l) + sumLeaves(r)
  case Leaf(x) => x
}
```

Scala – Higher-Order Functions

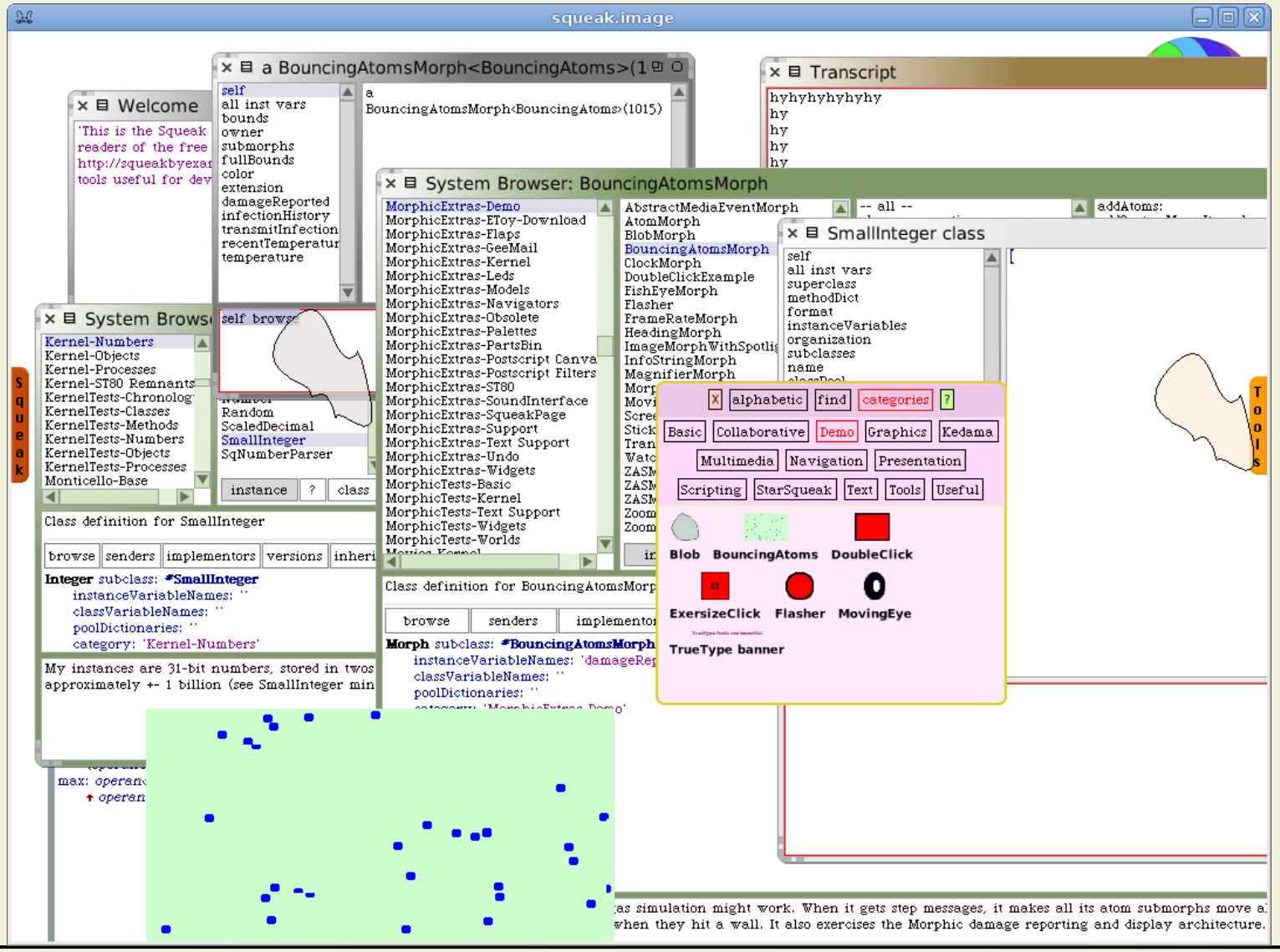
```
class Decorator(left: String,right: String)
{
  def layout[A](x: A) =
    left + x.toString() + right
}

object FunTest extends Application {
  def apply(f: Int=>String, v: Int) = f(v)
  val decorator = new Decorator("[", "]")
  println(apply(decorator.layout, 7))
}
```

Squeak

- “Squeak is a highly portable, open-source Smalltalk with powerful multimedia facilities. Squeak is the vehicle for a wide range of projects from educational platforms to commercial web application development.”
- “Squeak is based on Smalltalk which was created more than 35 years ago. Smalltalk defined the term object orientation and is the first language in which everything is built from objects.”

- `http://squeak.org/`

Squeak



C++

- ISO/IEC 14882 Standard from 1998
- Originally developed by Bjarne Stroustrup

"C++ is a general purpose programming language with a bias towards systems programming that

- is a better C
- supports data abstraction
- supports object-oriented programming
- supports generic programming."

Bjarne Stroustrup

Hello World in C++

```
#include <iostream>

int main() {
    std::cout << "Hello, world!\n";
}
```

Some Features of C++

- Everything that C has
- Adds OO features
- Operator overloading
- Flexible template mechanism
- Large Standard Template Library STL
- Huge library support, though not uniform

Inheritance and Method Binding in C++

- Multiple Inheritance
- Change of visibility of inherited classes
- Static method binding by default
- Dynamic method binding with `virtual` keyword
- Abstract classes by using pure virtual functions
- Method overriding and overloading

Example for C++

```
class Super {
   public:
      virtual void m() = 0;
}

class Sub : public Super, private Impl,
  public IFace {
   public:
      void m();
}
void Sub::m() { cout << "Sub::m\n"; }
```

C++ References

http://www.research.att.com/~bs/

http://www.open-std.org/jtc1/sc22/wg21/

C#

“C# (pronounced "C Sharp") is a simple, modern, object oriented, and type-safe programming language.
It will immediately be familiar to C and C++ programmers.
C# combines the high productivity of Rapid Application Development (RAD) languages and the raw power of C++.“

ECMA-334 C# Language Specification

Hello World in C#

```
using System;

class Hello {
    static void Main() {
      Console.WriteLine("hello, world");
    }
}
```

Some Features of C#

Basically everything from Java, plus:

- Definable value types `struct` and `enum`
- Four kinds of parameters: value, reference (`ref` keyword), and output parameters (`out` keyword), and parameter arrays (`params` keyword)
- Properties
- Indexers
- Operator overloading
- Delegates & Events
- Unsafe code sections

Array types: Rectangular or Jagged

```
class Test {
 static void Main() {
    int[] a1;   // single-dim array of int
    int[,] a2;  // 2-dimensional array of int
    int[,,] a3; // 3-dimensional array of int
    int[][] j2; // "jagged" array: array of
                // (array of int)
    int[][][] j3; // array of (array of
                  // (array of int))
  }
}
```

Inheritance and Method Binding in C#

- Single implementation inheritance
- Multiple interface subtyping
- Static method binding by default
- Dynamic method binding with `virtual` keyword
- Abstract classes and methods by using `abstract` keyword
- Keywords `override` and `new` distinguish between method overriding or hiding

Example for C#

```
using System;

public abstract class Super {
   public abstract void m();
}

public class Sub : Super, IFace {
   public [override | new] void m() {
      Console.WriteLine("Sub.m");
   }
}
```

C# References

http://msdn.microsoft.com/library/
default.asp?url=/library/en-us/cscon/
html/vcoricstartpage.asp

(base URL) + vclrfaquicksurveyof
(one word) csharpfeatures_pg.asp

http://genamics.com/developer/
csharp_comparative.htm

http://www.go-mono.com/

Questions?