

***Software
Technology
Group***

TU Darmstadt | FB Informatik

Software Engineering Design

Multi-Dispatch Lab

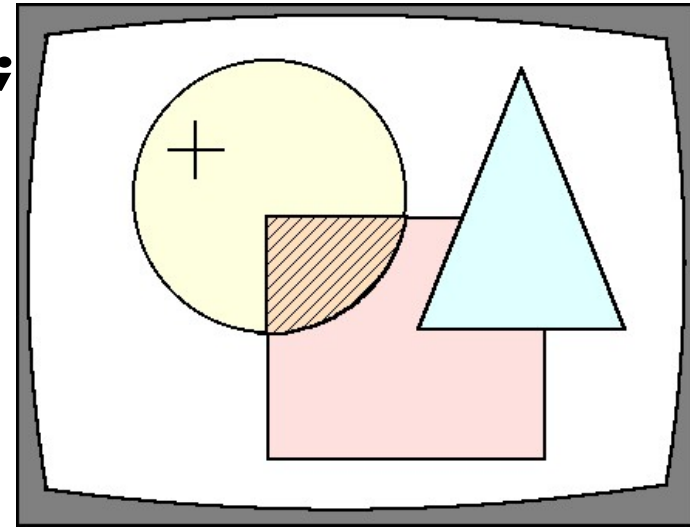
Prof. Dr. Mira Mezini

Dipl.-Inform. Christoph Bockisch

Dipl.-Ing. Michael Haupt

Remember: Double Polymorphism

```
Shape s1 = new Rectangle();  
Shape s2 = new Triangle();  
Shape s3 = new Circle();
```



```
...  
if (s1.intersect(s2))  
...  
...
```

We get `boolean Rectangle.intersect(???)`

One Solution: Double Dispatch

- We solved this problem by using double dispatch
- We already know there are languages supporting multi dispatch
- But why again is this useful?

LabA 1

- LabA1
 - Extend the Circle class to give more accurate messages like “intersecting Circle with Rectangle”
 - Use standard Java
 - At first use instanceof checks

One step towards multi methods

- It is ugly to have such tests in a method
- Can't the language choose an implementation for us?
- LabA2
 - Extend the Circle class to give more accurate messages like “intersecting Circle with Rectangle”
 - Use standard Java
 - Now use method overloading

Multi methods

- Java can only choose overloaded methods statically
- Multi methods are dispatched at runtime
- Like method overloading with extended syntax
 - The formal parameter type is now `<SuperType> @<ConcreteType>`
- LabA3
 - Extend the Circle class to give more accurate messages like “intersecting Circle with Rectangle”
 - Finally use multi methods
 - Invoke MultiJava compiler:
`java org.multijava.mjc.Main`
 - It must be Java 1.4

Multi methods

- Multi methods are always dispatched in a way that the method with the most specific formal parameters is selected
- The method with the next less specific parameters can be called by “resend()”
- To do so, the formal parameters must be final
boolean doesIntersect(final Shape@Circle s) { ... }
- super.<method>() calls the implementation in the super class
- Lab A4
 - Write different combinations of super / resend to find out how it works

Open classes

- We know open classes from AspectJ
- MultiJava also supports open classes
- Can use open classes to improve modularity when data and functionality change
- Syntax
 - When we want to add the method `prettyPrint()` to several types
 - Write “`String <Type>.prettyPrint() { ... }`” for all types
 - The file containing this is names “`prettyPrint.java`”
 - To call the method simply write “`<var of Type>.prettyPrint();`”

LabB1

- Lab B1
 - Add a pretty print method for all expressions
 - Add getter methods for left and right in BinaryExpression
 - Add getter method for nested expression in UnaryExpression
 - Extend the main method to invoke prettyPrint() for the generated expression

Lab B2

- Lab B2
 - Now add a new unary expression “Reciprocal”
 - Execute the program – what happens?