

[Home](#)[Newsletter](#)[Java Specialist Club](#)[Java Training](#)[Conference Venue Hire](#)[Java Resources](#)[Contact](#)

The Java Specialists' Newsletter

➤ Issue 009 ➤ 2001-02-15 ➤ Category: **Language** ➤ Java version:

0

[Subscribe Free](#) [RSS Feed](#) [Print](#)

Depth-first Polymorphism

by Dr. Heinz M. Kabutz

Welcome to the 9th issue of "The Java(tm) Specialists' Newsletter", where we look at "in-the-veld" tips and tricks used by Java professionals. I want to thank all of you who respond to these letters, your comments make the late nights writing these newsletters worthwhile :) And those of you who in their day-to-day communication are imitating the style of these newsletters - you know who you are ! - keep trying :-)

I must apologize that this newsletter is quite long. Unfortunately I am under severe time pressure at the moment so I did not have time to make it shorter.

I was told after last week's newsletter that Java was not based only on C++ and Smalltalk, but also on Lisp ?? (I did not know that, no wonder it's so terribly slow. I'm surprised the JDK doesn't come with the Emacs editor.) Anyway, I have to thank Michael Wolber from Infor AG in Germany for pointing that out and for his excellent contribution:

Greenspun's Tenth Rule of Programming: "any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp."

And Java?

Upcoming Java Master Courses:

Duesseldorf, Germany, Aug 22
Chania, Crete, Sep 6
Cape Town, South Africa, Sep 12

In-house courses if these dates or locations do not suit you. Note that the course in Crete may also be attended remotely via webinar.

Depth-first Polymorphism (or Customised Polyseme)

Consider the following class:

```
public class Polyseme {
    public static class Top {
        public void f(Object o) {
            System.out.println("Top.f(Object)");
        }
        public void f(String s) {
            System.out.println("Top.f(String)");
        }
    }
    public static void main(String[] args) {
        Top top = new Top();
        top.f(new java.util.Vector());
        top.f("hello");
        top.f((Object)"bye");
    }
}
```

Java looks for the method with the "narrowest" matching class for the parameter objects. Therefore, the output from running this class is:

```
Top.f(Object)
Top.f(String)
Top.f(Object)
```

In Java, the virtual machine tries to find a matching method for your parameters, starting at the top of the hierarchy and moving down. Say we have the following classes:

```
public class BreadthFirst {
    public static class Top {
        public void f(Object o) {
            System.out.println("Top.f(Object)");
        }
    }
    public static class Middle extends Top {
        public void f(String s) {
            System.out.println("Middle.f(String)");
        }
    }
}
```

Newsletter Links

- [Book Review](#)
- [Concurrency](#)
- [Exceptions](#)
- [GUI](#)
- [Inspirational](#)
- [Language](#)
- [Performance](#)
- [Software Engineering](#)
- [Tips and Tricks](#)

Java Courses

- Java Master
- Java Foundation
- Java 5 Tiger
- Design Patterns

[find out more](#)

What is the Java Specialists Club?

Venue for Hire

Looking for a super conference room for your next company event?

Crete is your perfect destination.

[Click here for more details](#)

```

    }
    public static void main(String[] args) {
        Top top = new Middle();
        top.f(new java.util.Vector());
        top.f("hello");
        top.f((Object)"bye");
    }
}

```

The virtual machine will thus start at Top and check if there are any methods which would accept String.class or Object.class, and indeed, Top.f(Object) would handle all those parameters. The output is therefore the following:

```

Top.f(Object)
Top.f(Object)
Top.f(Object)

```

We could "fix" this by overriding f(Object) and using instanceof to call the correct f() method (brrr - I'd rather get stuck on the N2 than do that [for those not living in Cape Town, the N2 is notoriously dangerous, you either get shot at or in or with if your car breaks down])

```

public class BreadthFirstFix {
    public static class Top {
        public void f(Object o) {
            System.out.println("Top.f(Object)");
        }
    }
    public static class Middle extends Top {
        public void f(Object o) {
            if (o instanceof String)
                f((String)o);
            else
                super.f(o);
        }
        public void f(String s) {
            System.out.println("Middle.f(String)");
        }
    }
    public static void main(String[] args) {
        Top top = new Middle();
        top.f(new java.util.Vector());
        top.f("hello");
        top.f((Object)"bye");
    }
}

```

The output would now look as we would expect:

```

Top.f(Object)
Middle.f(String)
Middle.f(String)

```

This might have the correct effect, but it does mean that we have to have such a silly "instanceof" in all the subclasses. If we are designing a OO framework we want to have our clients subclass our classes without having to do acrobatics to achieve this.

Christoph Jung mentioned this problem with Java to me a few weeks ago and we thought of some code you could put at the highest level class that uses reflection to start at the lowest class and then tries to match the method to the type before moving up the hierarchy. I call this "depth-first-polymorphism".

```

import java.lang.reflect.*;
public class DepthFirst {
    public static class Top {
        private Method getPolymorphicMethod(Object param) {
            try {
                Class c1 = getClass(); // the bottom-most class
                // we start at the bottom and work our way up
                Class[] paramTypes = {param.getClass()};
                while (!c1.equals(Top.class)) {
                    try {
                        // this way we find the actual method
                        return c1.getDeclaredMethod("f", paramTypes);
                    } catch (NoSuchMethodException ex) {}
                    c1 = c1.getSuperclass();
                }
                return null;
            }
            catch (RuntimeException ex) { throw ex; }
            catch (Exception ex) { return null; }
        }
        public void f(Object object) {
            Method downPolymorphic = getPolymorphicMethod(object);
            if (downPolymorphic == null) {
                System.out.println("Top.f(Object)");
            }
            else {
                try {
                    downPolymorphic.invoke(this, new Object[] {object});
                }
                catch (RuntimeException ex) { throw ex; }
                catch (Exception ex) {
                    throw new RuntimeException(ex.toString());
                }
            }
        }
    }
    public static class Middle extends Top {
        public void f(String s) {
            System.out.println("Middle.f(String)");
        }
    }
    public static class Bottom extends Middle {
        public void f(Integer i) {
            System.out.println("Bottom.f(Integer)");
        }
    }
    public static class RockBottom extends Bottom {

```

```
    public void f(String s) {
        System.out.println("RockBottom.f(String)");
    }
}
public static void main(String[] args) {
    Top top = new RockBottom();
    top.f(new java.util.Vector());
    top.f("hello");
    top.f(new Integer(42));
    top = new Bottom();
    top.f(new java.util.Vector());
    top.f("hello");
    top.f(new Integer(42));
}
}
```

The answer is this time:

```
Top.f(Object)
RockBottom.f(String)
Bottom.f(Integer)
Top.f(Object)
Middle.f(String)
Bottom.f(Integer)
```

When should you use this technique? Only if you have a lot of specific type handlers as subclasses of a common superclass where it would make sense to add such a depth-first invoker. You can probably extract this functionality and put it in a separate class. If you use this commercially please do the exception handling correctly, I didn't bother in my example, in preparation for when I change my logo to "The C# Specialists".

Thanks for your comments, I always appreciate your feedback.

Regards

Heinz

[▶ Language Articles](#) [▶ Related Java Course](#) [▶ Discuss at The Java Specialist Club](#)

© 2010 Heinz Kabutz - All Rights Reserved

[Sitemap](#)

[seo web design](#) Catch22 Marketing

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. JavaSpecialists.eu is not connected to Oracle, Inc. and is not sponsored by Oracle, Inc.