Design Patterns
The Timeless Way of Coding

Designed and Presented by

Dr. Heinz Kabutz

Illustrations by Edith Sher

Dr. Heinz Kabutz

- Professional Java Programmer
- Did PhD in Computer Science at the University of Cape Town, South Africa
- Trainer of Java and Design Patterns Courses in various places of the world
- Publish **The Java Specialists' Newsletter**
  - Only publication of its kind
  - Sent to over 100 countries
  - Archive on http://www.javaspecialists.co.za

Questions

- Please please please please ask questions!
- There are some stupid questions
  - They are the ones you didn't ask
  - Once you've asked them, they are not stupid anymore
- Assume that if you didn't understand something that it was my fault
- The more you ask, the more everyone learns (including me)

Structure of Talk

- Software Engineering
  - as it happens in the software factories
- How Design Patterns fit in
- Proxy Design Patterns
  - Demonstration: Virtual and security proxies with Java dynamic proxies
- Singleton Misunderstood (time permitting)
  - Common misconceptions with Singleton
- Discussion time

1. Software Engineering

- Why do companies want experience?
- What experience is most valuable?
- Experience in which language will guarantee you a job?

Classic Methodologies

- e.g. Waterfall Model: Analysis, Design, Implementation, Testing
- Suffered from “Analysis Paralysis”
- Bad decision during analysis very expensive
- Nice model for large teams with greatly varying skill-sets
- Each iteration takes months

Agile Methodologies

- e.g. eXtreme Programming
- All programming is done in pairs
  - For constant code reviewing, NOT mentoring
- Very short iterations (days or even hours)
- Testing is done several times a day
- Daily automated build and complete test
- Designing with Patterns
- Ruthless refactoring

Which Methodology to Use?

- Waterfall Model
  - One or two excellent analysts
  - Few good designers
  - Lots of average programmers
  - Suffers from “Peter Principle”
- eXtreme Programming
  - Between 6 and 12 **above** average programmers per team
  - Fosters cooperation, not competition in team
  - Low staff turnover
  - Chaos if not strictly managed!!!

Typical Day as Programmer

08:00 Arrive at work

08:30 Had first cup of coffee, erased SPAM

09:00 Chatted with coworker about soccer

10:00 Had project status meeting

11:00 Thought about design problems (Whilst playing minesweeper)

12:30 Looked at some critical bugs for important customer

13:30 Finished playing “Battlefield 1942” with colleagues

15:00 Wrote 200 lines of VB code, answered 5 phone calls

16:30 Company meeting entitled “Be more productive”

17:30 Wrote emails to bosses and colleagues (and friends)

23:30 Time to go home – finished writing TCP/IP stack in assembler

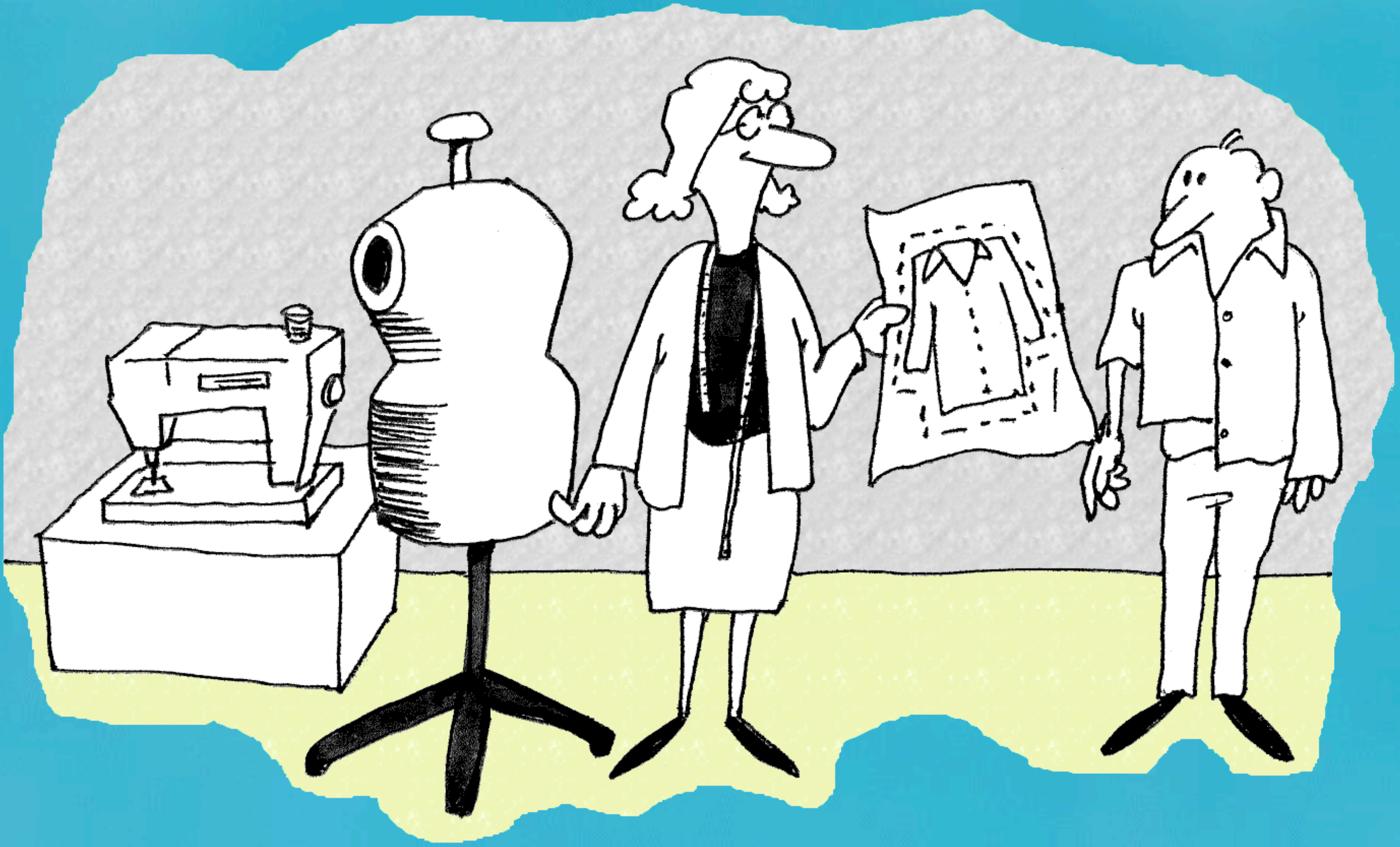
Programming is a Minority Task

- Most of your time is spent in:
  - Meetings
  - Documentation
  - Planning
  - Testing, bug fixing & support
  - Email
- Even brilliant programmers have to communicate!

Design Language can Help

- Meetings
  - Communicate more effectively about your designs to colleagues
- Documentation
  - Code documentation can refer to Design Pattern
- Planning
  - You can talk in higher-level components
- Testing, bug fixing & support
  - Better designs will reduce bugs and make code easier to change

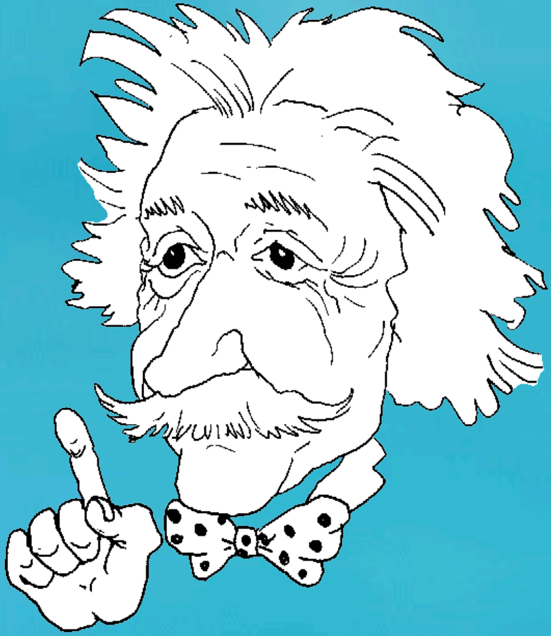
2: Introduction to Patterns



Vintage Wines

- Design Patterns are like good red wine
  - You cannot appreciate them at first
  - As you study them you learn the difference between table wine and vintage
  - As you become a connoisseur you experience the various textures you didn't notice before
- Warning: Once you are hooked, you will no longer be happy with plain table wine!

Why are patterns so important?



- Provide a view into the brains of OO experts
- Help you understand existing designs
- Patterns in Java, Volume 1, Mark Grand writes
  - "What makes a bright, experienced programmer much more productive than a bright, but inexperienced, programmer is experience."

Design Patterns Origin

The Timeless Way of Building

Christopher Alexander

There is a central quality which is the root criterion of life and spirit in a man, a town, a building, or a wilderness.

If you want to make a living flower, you don't build it physically, with tweezers, cell by cell. You grow it from the seed.

Textbook – "Design Patterns"

- "Design Patterns" book by Gang of Four (GoF)
- Contains a collection of basic "patterns" that experienced OO developers use regularly
- Cannot proceed very far in Java, C#, VB.NET without understanding patterns
- Facilitates better communication
- Based on work of renegade architect Christopher Alexander in "The Timeless Way of Building"

What’s in a name?

The Timeless Way of Building

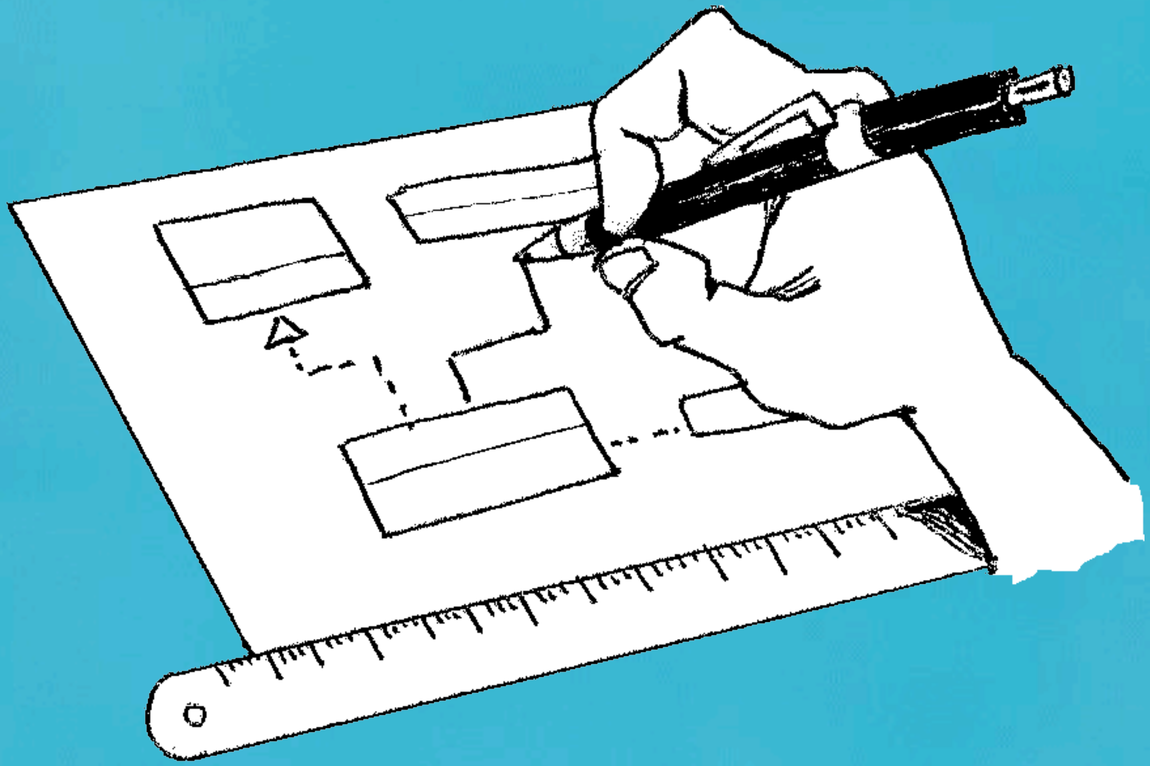
The search for a name is a fundamental part of the process of inventing or discovering a pattern.

So long as a pattern has a weak name, it means that it is not a clear concept, and you cannot tell me to make “one”.

Why do we need a diagram?

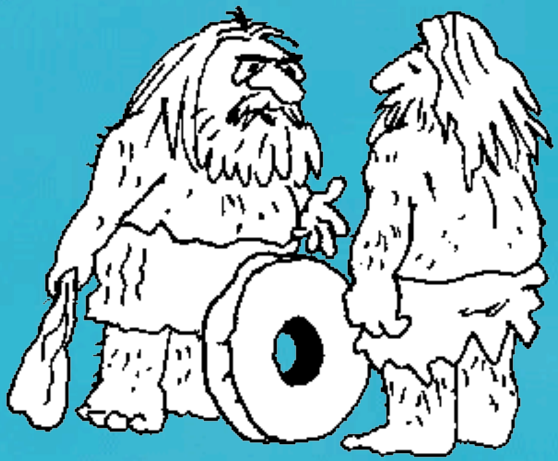
The Timeless Way of Building

If you can't draw a [class] diagram of it, it isn't a pattern



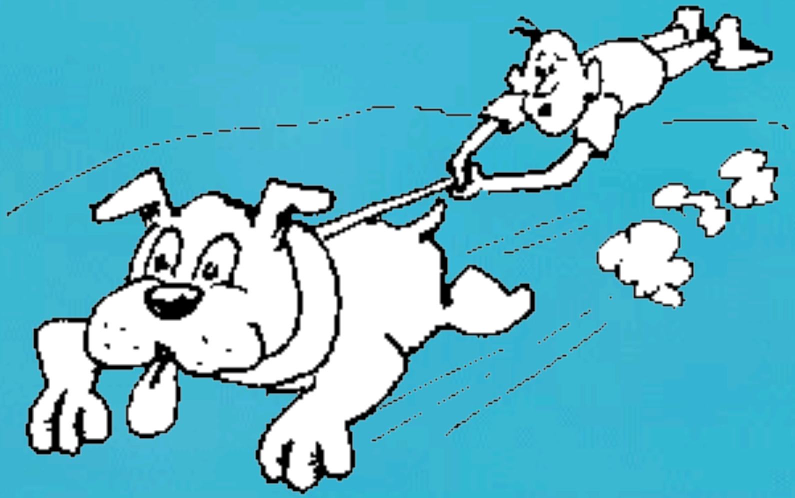
Misuse of Design Patterns

- Patterns Misapplied
  - “design” patterns should not be used during analysis
- Cookie Cutter Patterns
  - patterns are generalised solutions
- Misuse By Omission
  - reinventing a crooked wheel

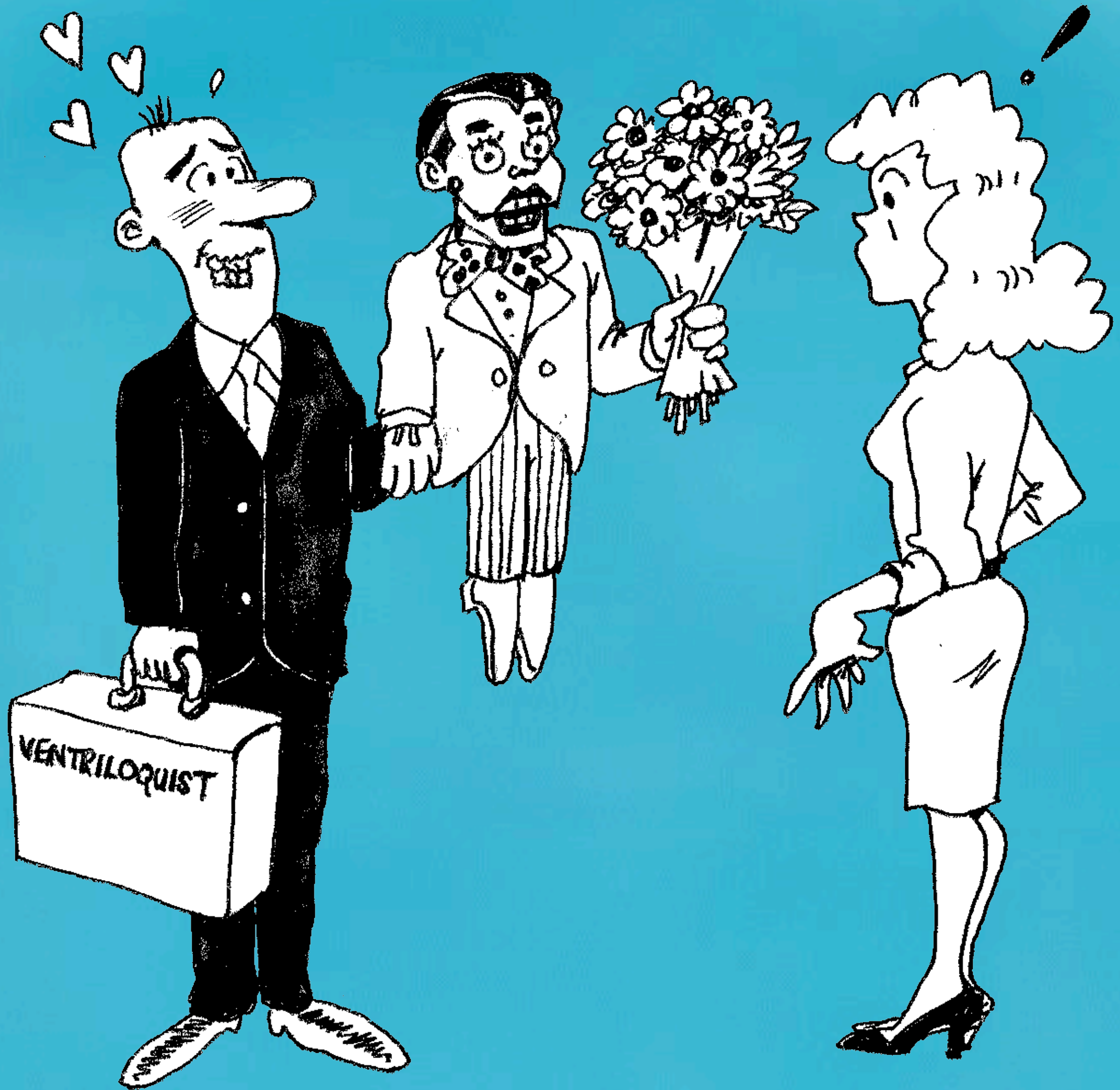


Summary

- Object Orientation is here to stay
- Design Patterns will fast-track you in learning how to design with objects

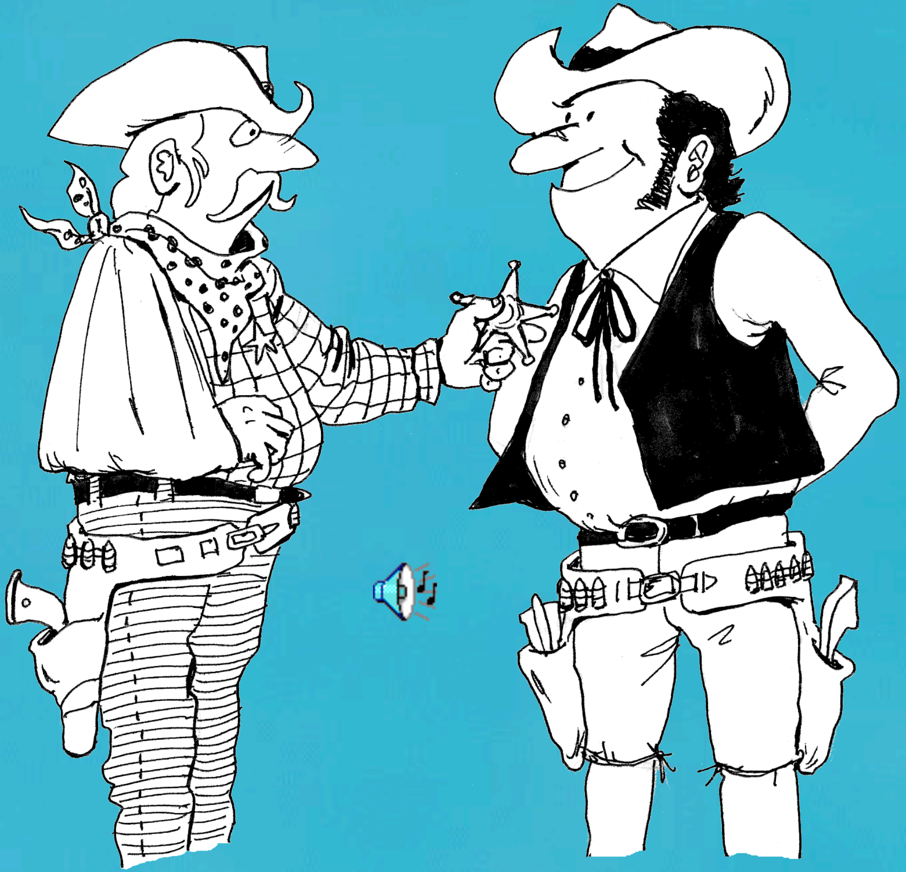


3: Proxy

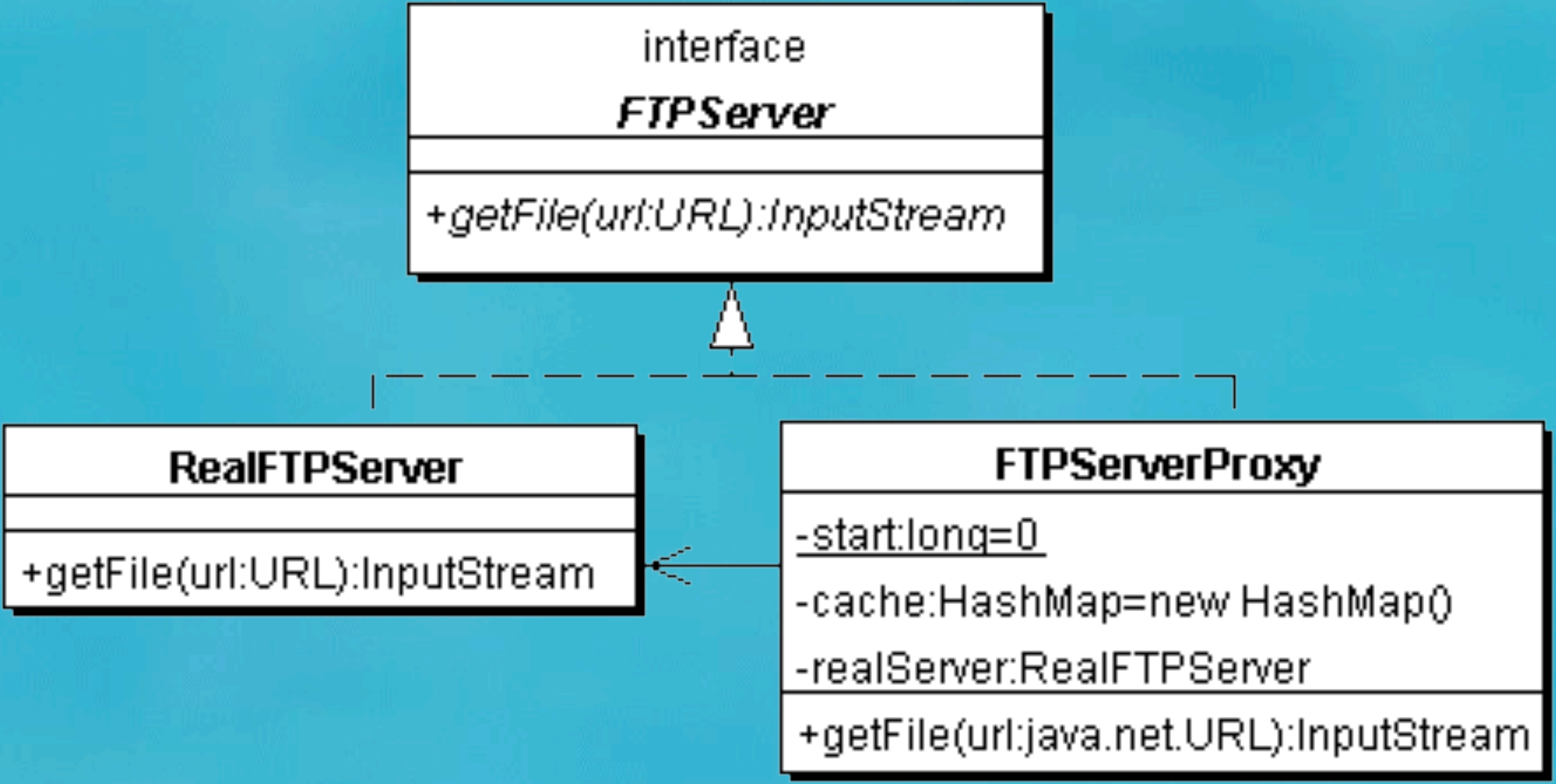


Proxy

- Intent
  - Provide a surrogate or placeholder for another object to control access to it.
- Also known as
  - Surrogate



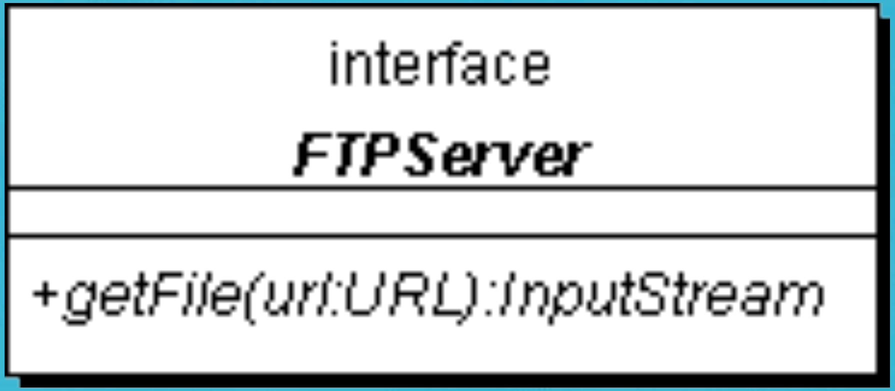
Motivation: Proxy


interface
FTPServer
+getFile(url:URL):InputStream
RealFTPServer
+getFile(url:URL):InputStream
FTPServerProxy
-start:long=0
-cache:HashMap=new HashMap()
-realServer:RealFTPServer
+getFile(url:java.net.URL):InputStream

FTPServer interface

- Defines a common method “getFile”

```
import java.net.URL;
import java.io.*;

public interface FTPServer {
  public InputStream getFile(URL url)
    throws IOException;
}
```



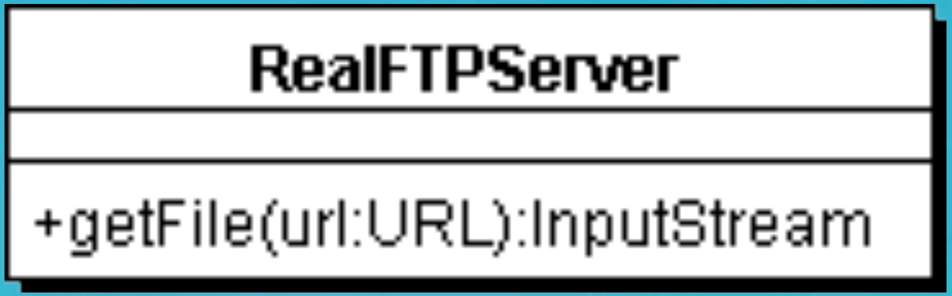
interface

FTPServer

+getFile(url:URL):InputStream

RealFTPServer

- Reading a file across the network
- Implements FTPServer interface



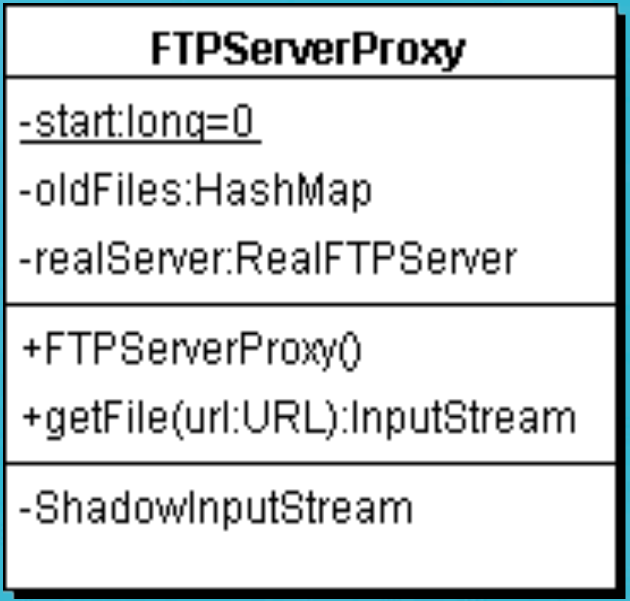
RealFTPServer

+getFile(url:URL):InputStream

```
import java.net.*;
import java.io.*;

public class RealFTPServer implements FTPServer {
  public InputStream getFile(URL url)
      throws IOException {
    System.out.println(
      "Getting file from real FTP Server");
    return url.openStream();
  }
}
```

FTPServerProxy

- Fetches files from RealFTPServer and writes them to disk with ShadowInputStream
- Next time the same file is requested it is returned directly
  from the disk
- Speeds up file retrieval



FTPServerProxy

-start:long=0

-oldFiles:HashMap

-realServer:RealFTPServer

+FTPServerProxy()

+getFile(url:URL):InputStream

-ShadowInputStream

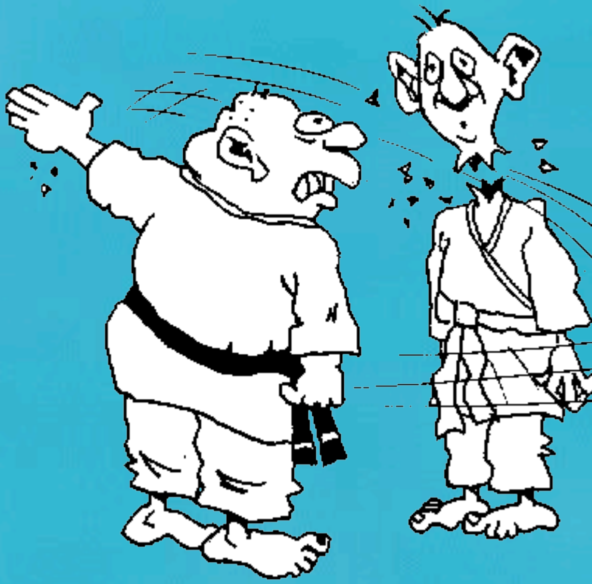
```
import java.util.*;
import java.io.*;
public class FTPServerProxy implements FTPServer {
  private static long cacheID = 0;
  private HashMap cache = new HashMap();
  private RealFTPServer realServer =
    new RealFTPServer();
  // write file to local disk as it gets read
  public InputStream getFile(java.net.URL url)
      throws IOException {
    if (cache.containsKey(url)) {
      System.out.println("Getting file from cache");
      return new FileInputStream(
        (String)cache.get(url));
    }
    String filename = cacheID++ + ".cache";
    cache.put(url, filename);
    return new ShadowInputStream(
      realServer.getFile(url),
        new FileOutputStream(filename));
  }
```

```java
// Copies the bytes read from the InputStream to the
// specified OutputStream
class ShadowInputStream extends FilterInputStream {
  private final OutputStream out;
  public ShadowInputStream(InputStream in,
      OutputStream out) {
    super(in);
    this.out = out;
  }
  public int read() throws IOException {
    int result = super.read();
    if (result != -1) out.write(result);
    return result;
  }
  public int read(byte[] buf, int offset, int length)
      throws IOException {
    int result = super.read(buf, offset, length);
    if (result != -1) out.write(buf, offset, result);
    return result;
  }
```

```
    public void close() throws IOException {
      super.close();
      out.close();
    }
  } // end of class ShadowInputStream
} // end of class FTPServerProxy
```

Not Robust Enough

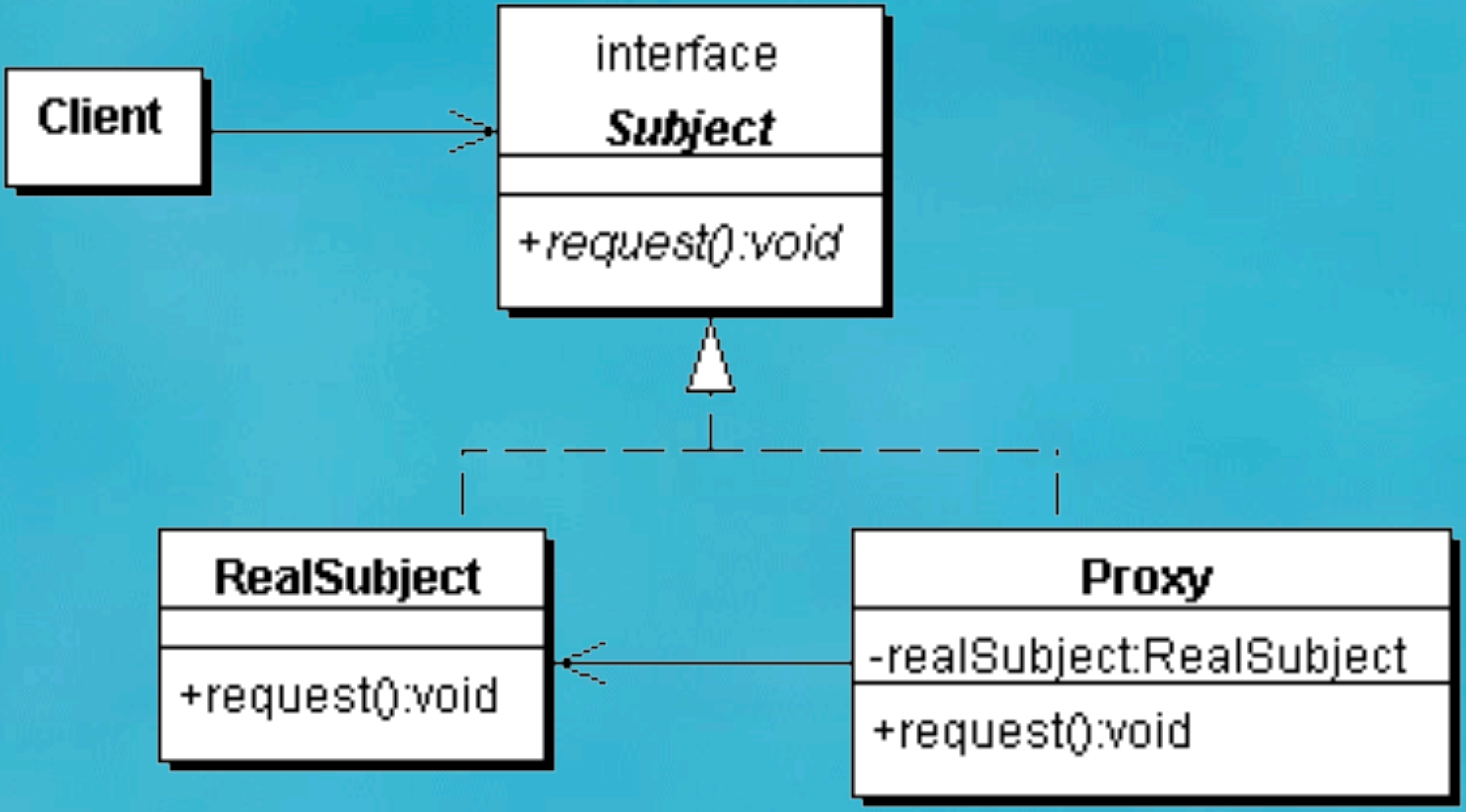
- There should be a facility for deleting older files
- Partial reads should be resumed
- Can’t handle simultaneous connections for some URL
- Versions of URLs
- Code was kept “simple”



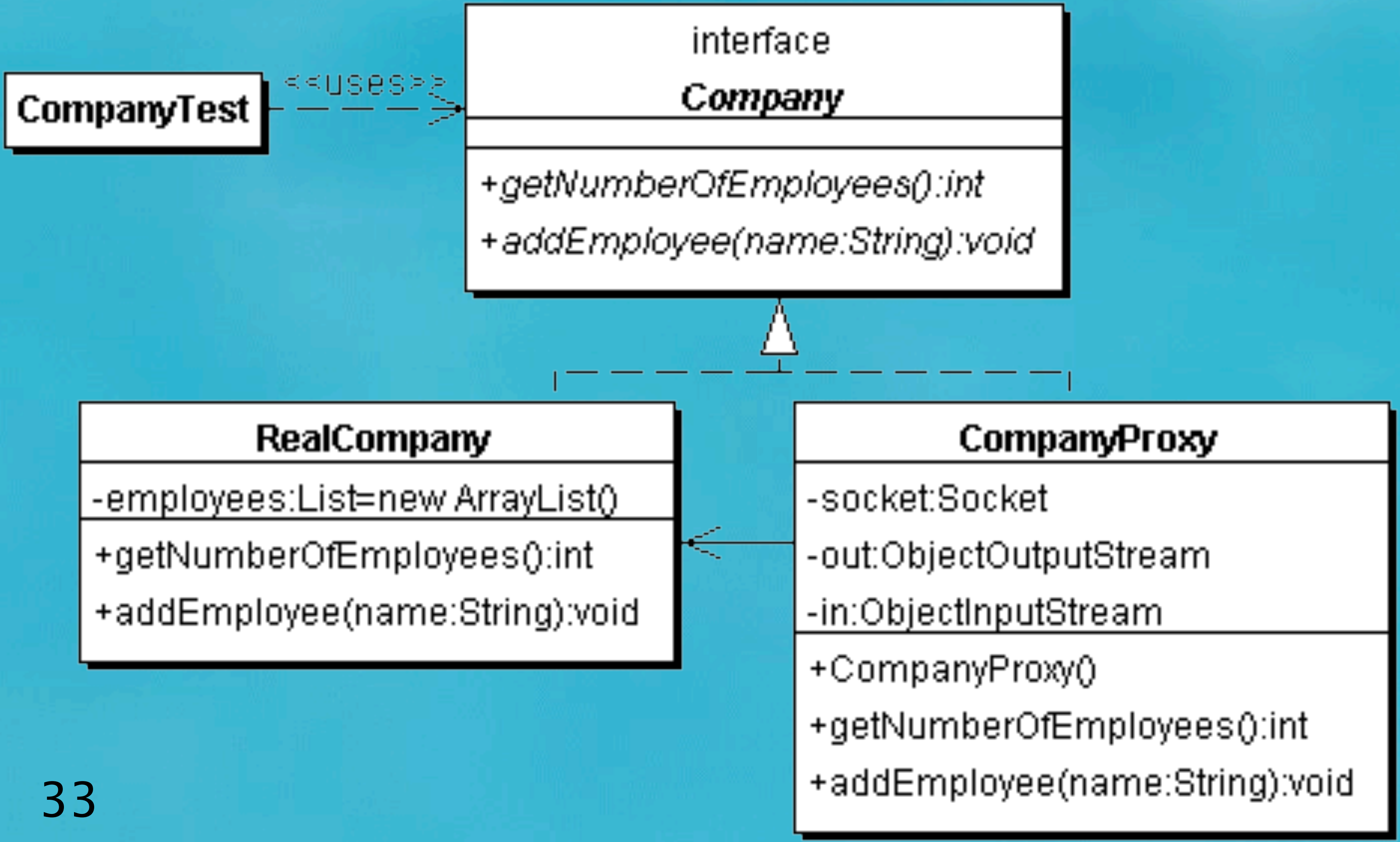
Applicability: Proxy

- Virtual Proxy
  - creates expensive objects on demand
- Remote Proxy
  - provides a local representation for an object in a different address space
- Protection Proxy
  - controls access to original object

Structure: Proxy



Remote Proxy


CompanyTest
<<uses>>
interface
Company
+getNumberOfEmployees():int
+addEmployee(name:String):void
RealCompany
-employees:List=new ArrayList()
+getNumberOfEmployees():int
+addEmployee(name:String):void
CompanyProxy
-socket:Socket
-out:ObjectOutputStream
-in:ObjectInputStream
+CompanyProxy()
+getNumberOfEmployees():int
+addEmployee(name:String):void

```
public interface Company {
  int getNumberOfEmployees();
  void addEmployee(String name);
}

// The “RealCompany” would typically be an object
// contained by an EJB container or RMI server
import java.util.*;
public class RealCompany implements Company {
  private List employees = new ArrayList();
  public int getNumberOfEmployees() {
    return employees.size();
  }
  public void addEmployee(String name) {
    employees.add(name);
  }
}
```

```java
public class CompanyProxy implements Company {
  // various data members for network comms
  public int getNumberOfEmployees() {
    // send a message over the network to fetch
    // the value from the RealCompany class
    // sitting on the server
  }
  public void addEmployee(String name) {
    // send a message over the network to create
    // an employee in the remote company
  }
  // etc.
}
```

```
public class CompanyTest {
  public static void main(String[] args)
      throws java.io.IOException {
    Company company = new CompanyProxy();
    company.addEmployee("John Smith");
    System.out.println("Now there are " +
      company.getNumberOfEmployees() +
      " employees");
  }
}
```

Our client code can talk to the **Company** as if it were a local object

Consequences: Proxy

- Introduces level of indirection when accessing an object
  - A remote proxy can hide the fact that an object resides in a different address space
  - A virtual proxy can perform optimizations such as creating an object on demand
- The proxy and the real subject are objects of different types
  - Make sure equals(Object) caters for this!
- Another optimization is copy-on-write
  - e.g. `java.lang.StringBuffer`

Known Uses in Java: Proxy

- Remote proxies created transparently by **rmic** tool for Remote Method Invocation (RMI) mechanism
- EJB deployment tools call **rmic** transparently
- JDK 1.3 has support for dynamic proxies
  - Can add a proxy to a live object
  - Covered in “The Java™ Specialists’ Newsletter”

Questions: Proxy

- You have designed a Java server that connects to a database. If several clients connect to your server at once, how could Proxies be of help?
- If a Proxy is used to control access to another object, does the Proxy simplify code?

Exercises: Proxy

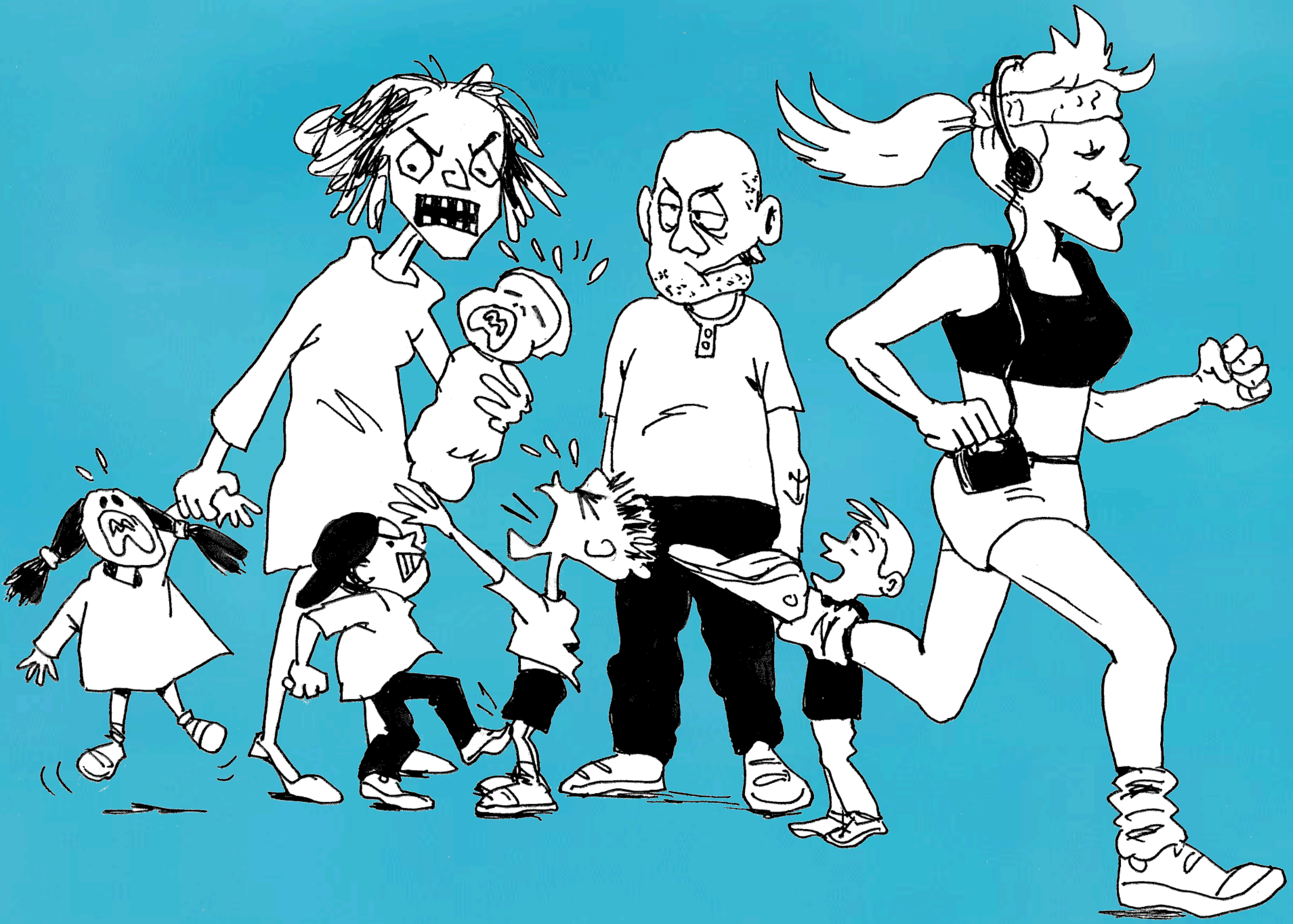
- Consider the following Employee classes. Write a SecureEmployee class which implements the Employee interface, but checks that the SecurityManager allows access to salaries. Test it against the EmployeeTest class.

```
public interface Employee { // the Subject
  /**@throws SecurityException if access denied */
  double getSalary();
}
public class RealEmployee implements Employee {
  public double getSalary() {
    return 321444.22;
  }
}
```

Exercises: Proxy

```
public class SecurityManager {
  private static boolean salary;
  public static void setSalaryPermission(
    boolean val) {salary = val;}
  public static void checkSalaryPermission() {
    if (!salary) throw new SecurityException();
  }
}
public class EmployeeTest {
  public static void main(String[] args) {
    Employee maxi = new SecureEmployee(
      new RealEmployee());
    SecurityManager.setSalaryPermission(true);
    System.out.println(maxi.getSalary());
    SecurityManager.setSalaryPermission(false);
    System.out.println(maxi.getSalary());
  }
}
```

4. Singleton

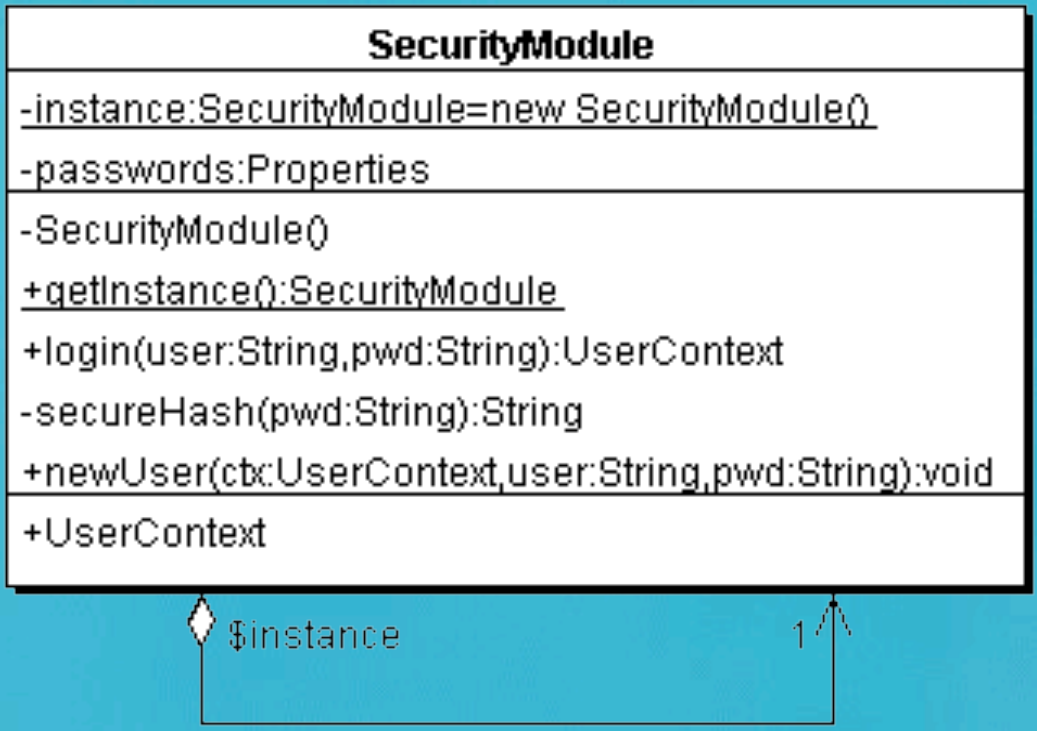


Singleton

- Intent
  - Ensure a class only has one instance, and provide a global point of access to it.

Motivation: Singleton

- It’s important for some classes to have exactly **one** instance, e.g. SecurityModule



SecurityModule

-instance:SecurityModule=new SecurityModule()

-passwords:Properties

-SecurityModule()

+getInstance():SecurityModule

+login(user:String,pwd:String):UserContext

-secureHash(pwd:String):String

+newUser(ctx:UserContext,user:String,pwd:String):void

+UserContext

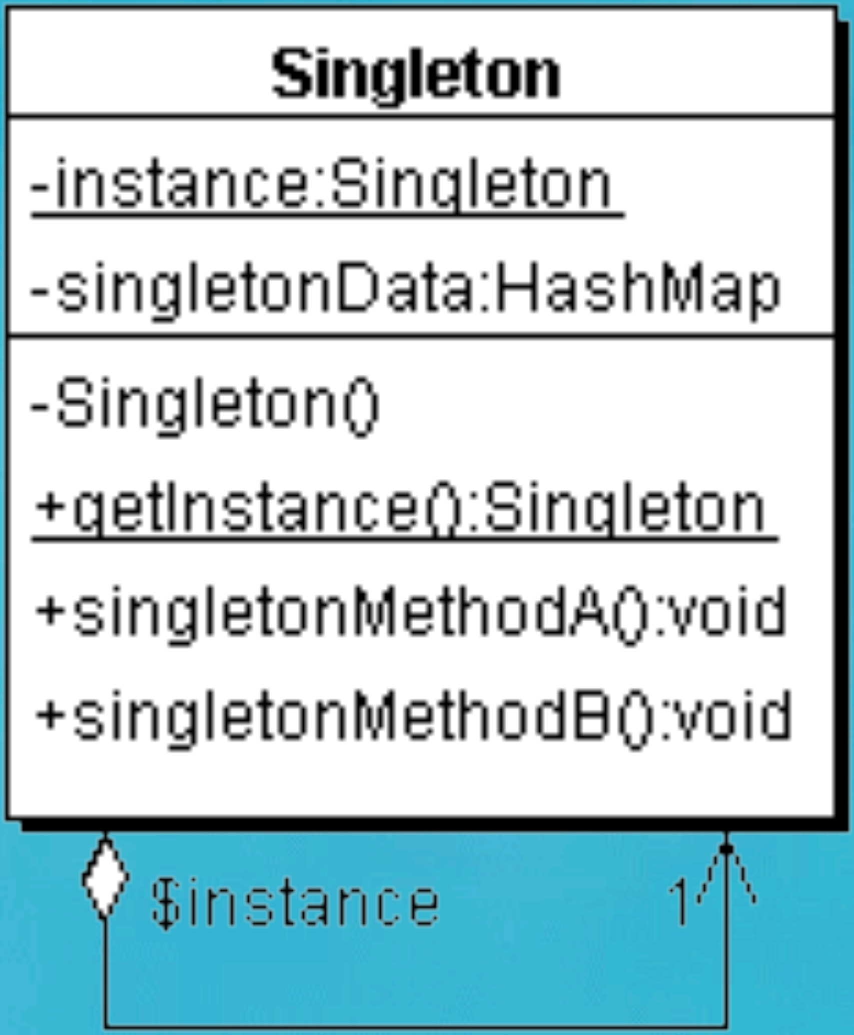
Sample Code: Singleton

```
public class SecurityModule {
  private static SecurityModule instance =
    new SecurityModule();

  public static SecurityModule getInstance() {
    return instance;
  }

  private SecurityModule() {
    loadPasswords();
  }

  public UserContext login(String username,
      String password) {
    return new UserContext(username, password);
  }

  // etc.
```

Applicability: Singleton

- Use the Singleton pattern when
  - there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
  - when the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

Structure: Singleton



Singleton

-instance:Singleton

-singletonData:HashMap

-Singleton()

+getInstance():Singleton

+singletonMethodA():void

+singletonMethodB():void

Consequences: Singleton

- Benefits
  - Controlled access to sole instance
  - Reduced name space
  - Permits refinement of operations and representation
  - Permits a variable number of instances
  - More flexible than class operations
- Drawbacks
  - Overuse can make a system less OO.

Known Uses in Java: Singleton

- java.lang.Runtime.getRuntime()
- java.awt.Toolkit.getDefaultToolkit()

Questions: Singleton

- The pattern for Singleton uses a private constructor, thus preventing extendability. What issues should you consider if you want to make the Singleton “polymorphic”?
- Sometimes a Singleton needs to be set up with certain data, such as filename, database URL, etc. How would you do this, and what are the issues involved?

Exercises: Singleton

- Turn the following class into a Singleton:

```
public class Earth {
  public static void spin() {}
  public static void warmUp() {}
}

public class EarthTest {
  public static void main(String[] args) {
    Earth.spin();
    Earth.warmUp();
  }
}
```

- Now change it to be extendible

5. Conclusion

- Software Engineering is essential for developing solid programs
- Architecture, Design, Performance all play a part

6. Some Thoughts

- Greece
  - Full member of the European Union
  - You can work in Germany, France, Netherlands, Belgium
- Entrepreneurship
  - Would you like to become a space tourist?
    - (Cost $ 20,000,000 for one flight)
  - Young means small expenses
  - Don’t work for someone – start your own companies!
    - Develop good products sold internationally

Body Shops

- Hundreds of underpaid programmers developing software for USA Germany UK
- Eastern Europe very cheap
  - $10 per hour
- India Wipro has 10,000 Java developers!
- Intel Science & Engineering Fair
  - Largest pre-university science competition
  - America: 65,000
  - China: 6,000,000
- **Greece cannot compete at that level!**

Innovation, Innovation

- Develop own ideas
- Start own companies
- Write the products, market them, sell them
- No one needs to know where software was produced
  - And no one cares either
- E.g. Thawte Consulting
  - Started in parent's garage, sold for $600,000,000
- E.g. The Java™ Specialists' Newsletter
  - Reaching over 100 countries!

Crete

- Crete is an excellent base from which to work
- Sunshine, beauty, friendly people
- Good quality of life
- Close to everywhere in Europe
- Access to internet
- Expenses low

Future

- We plan to move to Crete in the near future God willing

The Java™ Specialists' Newsletter

Produced on the beautiful island of Crete

- See you soon!

Design Patterns Cape Town

Design Patterns Germany

Design Patterns London

Design Patterns Switzerland

Design Patterns Estonia
at –18º Celsius

Design Patterns
Mauritius 2001

Tsinghua China 2003

华大学国际精品课程—面向对象的软件分析与设计高级研修
inghua University International Prime Course-Object-Oriented Software Analysis and Design Advance semi
2003.9.22-2003.9.

Austria
2005