Design Patterns

The Timeless Way of Coding

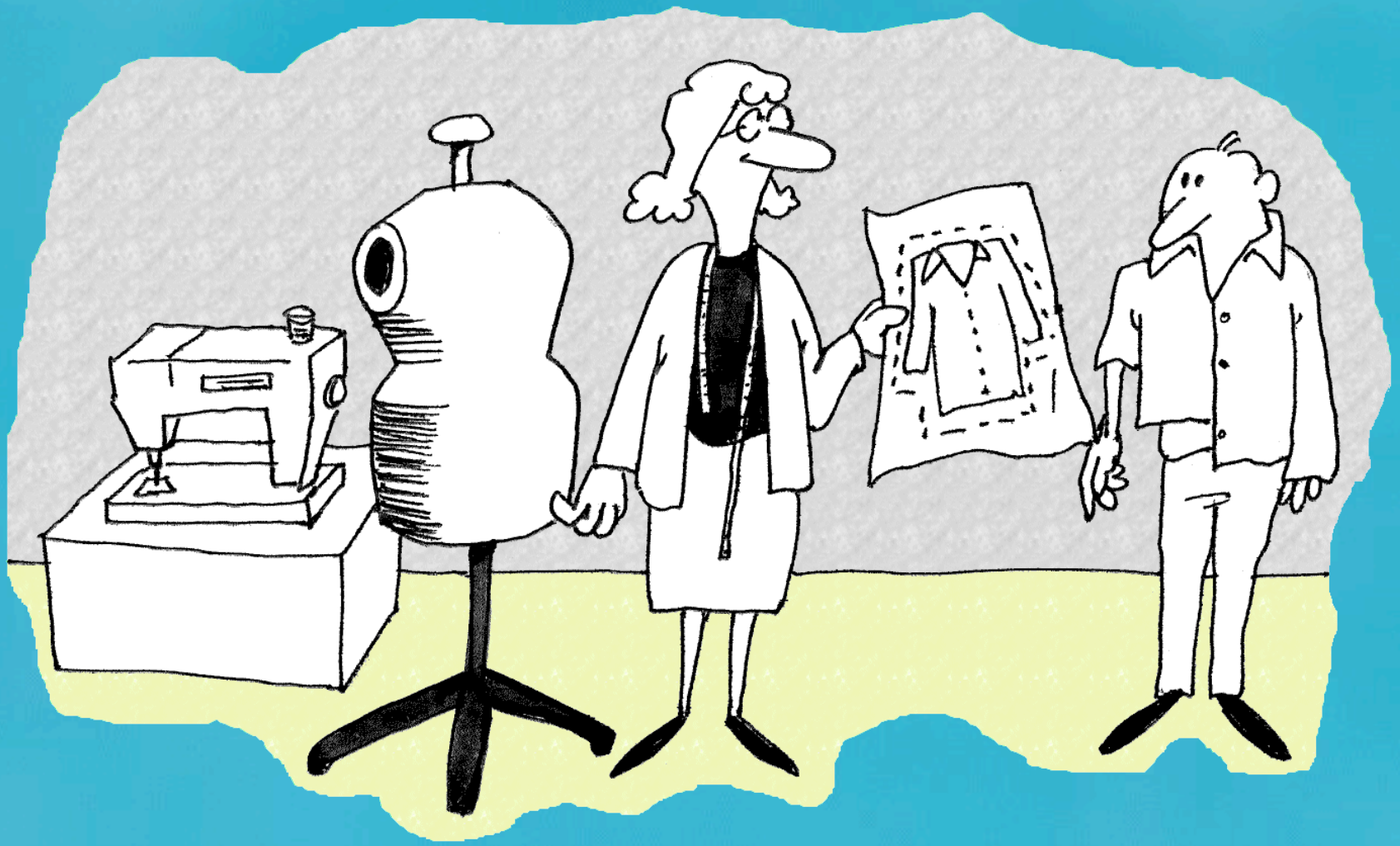
Designed and Presented by

Dr. Heinz Kabutz

Illustrations by Edith Sher

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

1: Introduction to Patterns



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Who am I?

- Java Programmer
- Trainer of Java and Design Patterns Courses in various places of the world
- Java Consultant
  - combining **Con**fuse and In**sult**
- Publish a Java newsletter with a small audience of Java specialists

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Purpose of Talk

1. To give something back to the Cape Town Java community
2. Explain a bit about why Design Patterns are important
3. Inspire you to learn for yourself

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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)

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

What is a Design Pattern?

- A design idea that has been applied many times, with success
- Designs that result in reusable code
- In our case, we will look at Object Oriented Design Patterns

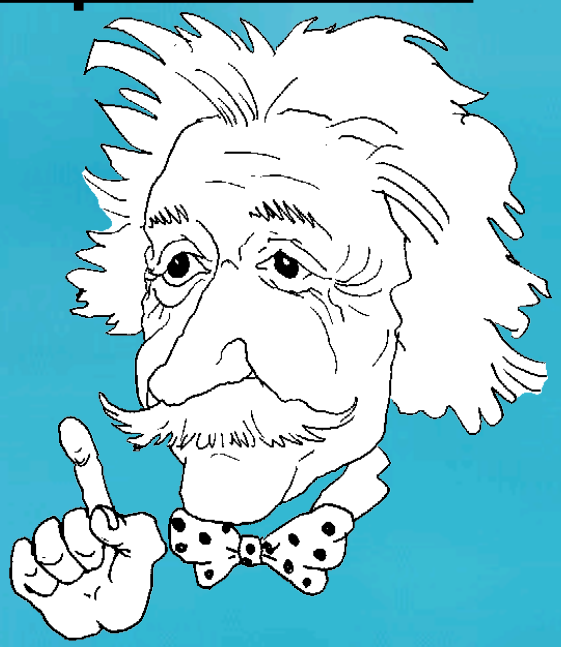
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Vintage Wines

- Design Patterns are like good red wine
  - You cannot appreciate them at first
  - As you study them you learn the difference between plonk 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 satisfied with plonk!

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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."

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Coding Patterns

- We have all seen patterns in code:
  - `for (int i=0; i<names.length; i++) ...`
  - common data structures, like linked list
- This is the way we “do things”
- Most courses teach the syntax of a language, not the semantics
- Design is normally learnt through experience

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Introduction

- For this talk I assume you have a good understanding of the basic OO concepts of encapsulation, abstraction, composition and inheritance
- Should be able to follow basic UML class diagrams
- Design Patterns is the recommended text; additional references are shown where applicable

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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”

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Pattern Structure

- Classic
  - Intent
  - Also Known As
  - Motivation
  - Applicability
  - Structure
  - Participants
  - Collaborations
  - Consequences
  - Implementation
  - Sample Code
  - Known Uses
  - Related Patterns
- This Course
  - Intent
  - Also Known As
  - Motivation
  - Sample Code
  - Applicability
  - Structure
  - Consequences
  - Known Uses In Java
- The other sections are left for self-study

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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.

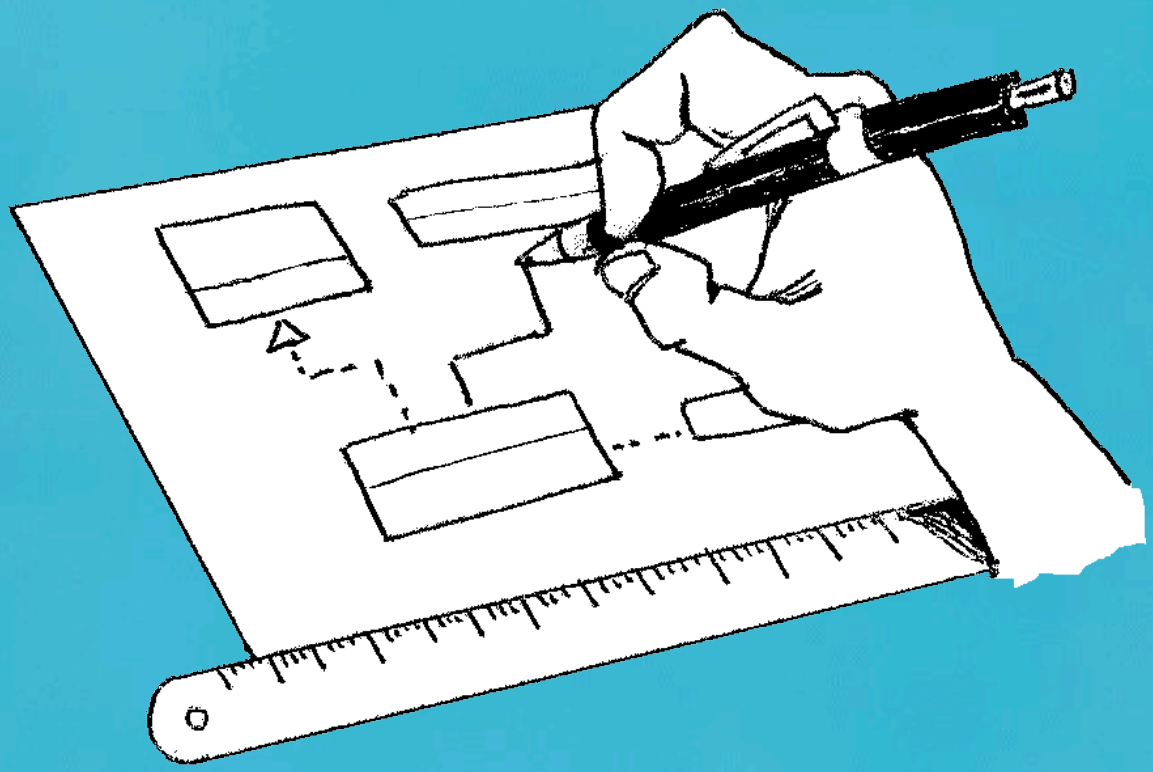
So long as a pattern has a weak name, it means that it is not a clear concept, and you cannot tell me to make "one".

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Why do we need a diagram?

The Timeless Way of Building

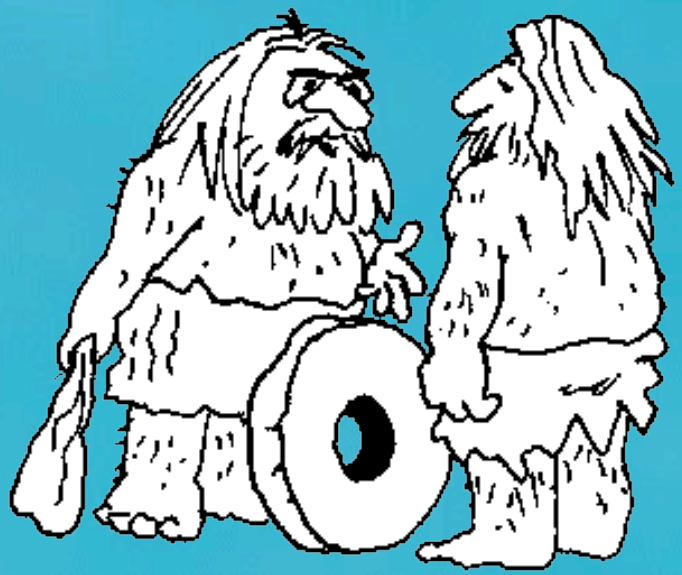
If you can't draw a [class] diagram of it, it isn't a pattern



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Misuse of Design Patterns

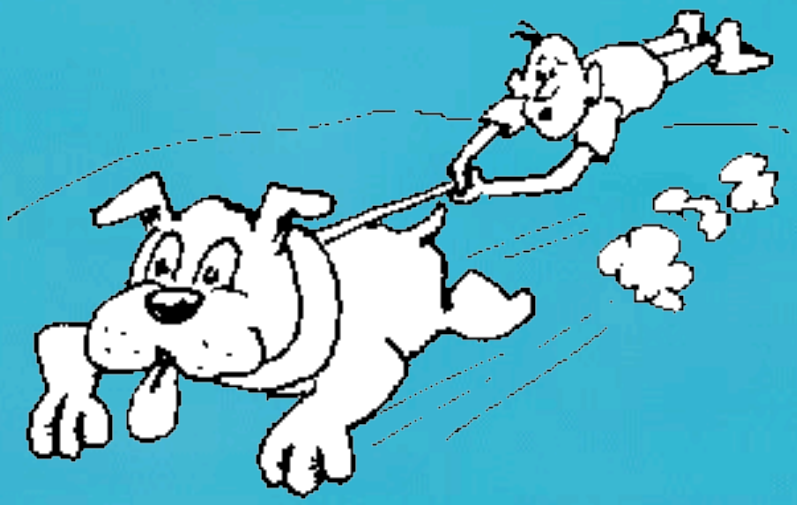
- Patterns Misapplied
  - "design" patterns should not be used during analysis
- Cookie Cutter Patterns
  - patterns are generalised solutions
- Misuse By Omission
  - reinventing a crooked wheel



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Summary

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



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

2: Composite

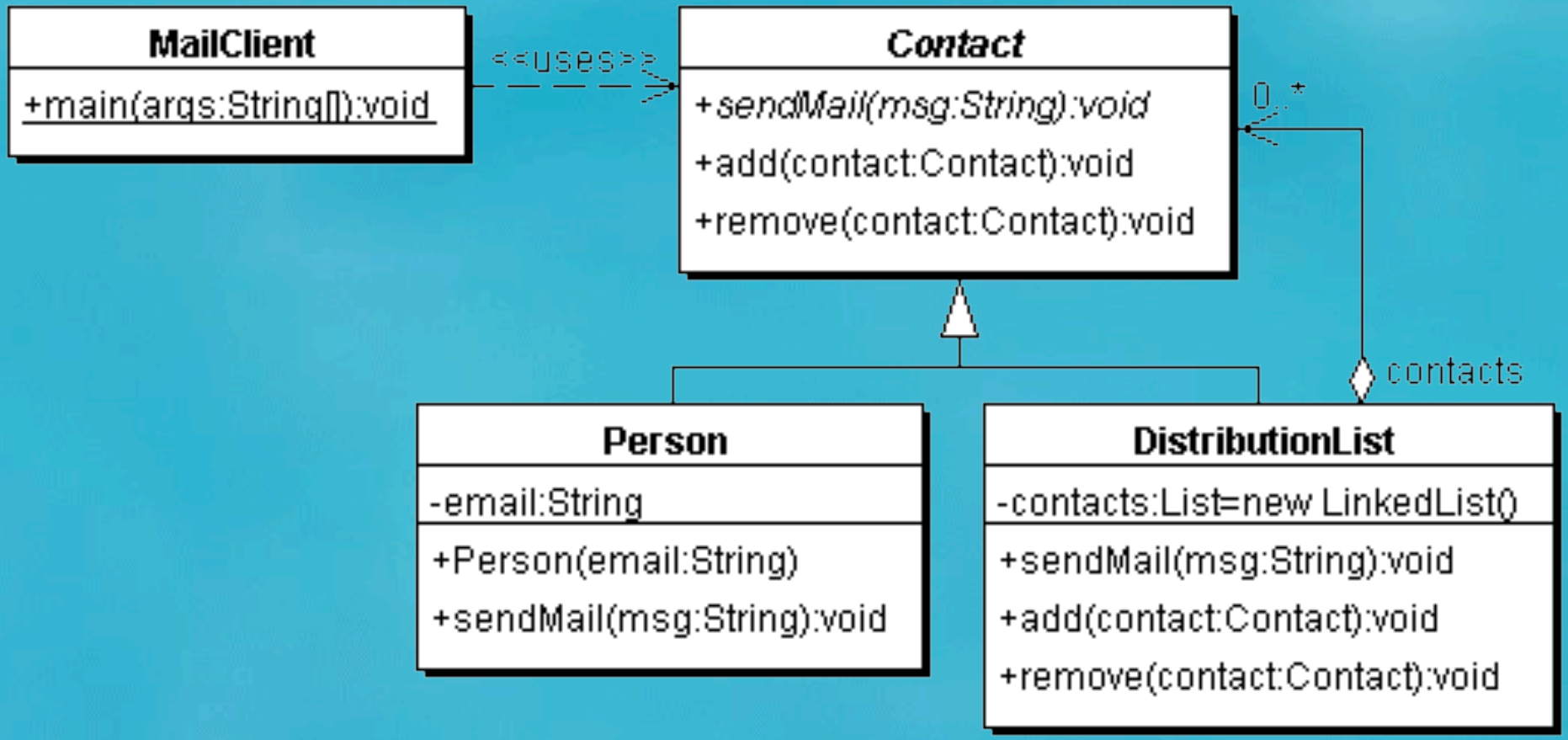
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Composite

- Intent
  - Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Intent according to Vlissides
  - Assemble objects into tree structures. Composite simplifies clients by letting them treat individual objects and assemblies of objects uniformly.

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Motivation: Composite



Sample Code: Contact

```
public abstract class Contact {
  public void add(Contact contact) {}
  public void remove(Contact contact) {}
  public abstract void sendMail(String msg);
}
```

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

Sample Code: Person

```
public class Person extends Contact {
  private final String email;
  public Person(String email) {
    this.email = email;
  }

  public void sendMail(String msg) {
    System.out.println("To: " + email);
    System.out.println("Msg: " + msg);
    System.out.println();
  }
}
```

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

Sample Code: DistributionList

```
import java.util.*;
public class DistributionList extends Contact {
  private List contacts = new LinkedList();
  public void add(Contact contact) {
    contacts.add(contact);
  }
  public void remove(Contact contact) {
    contacts.remove(contact);
  }

  public void sendMail(String msg) {
    Iterator it = contacts.iterator();
    while(it.hasNext()) {
      ((Contact)it.next()).sendMail(msg);
    }
  }
}
```

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

Sample Code: MailClient

```
public class MailClient {
  public static void main(String[] args) {
    Contact tjsn = new DistributionList();
    tjsn.add(new Person("john@aol.com"));
    Contact students = new DistributionList();
    students.add(new Person("amrita@intnet.mu"));
    tjsn.add(students);
    tjsn.add(new Person("anton@bea.com"));
    tjsn.sendMail(
      "welcome to the 5th edition of ...");
  }
}
```

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

> java MailClient ↵

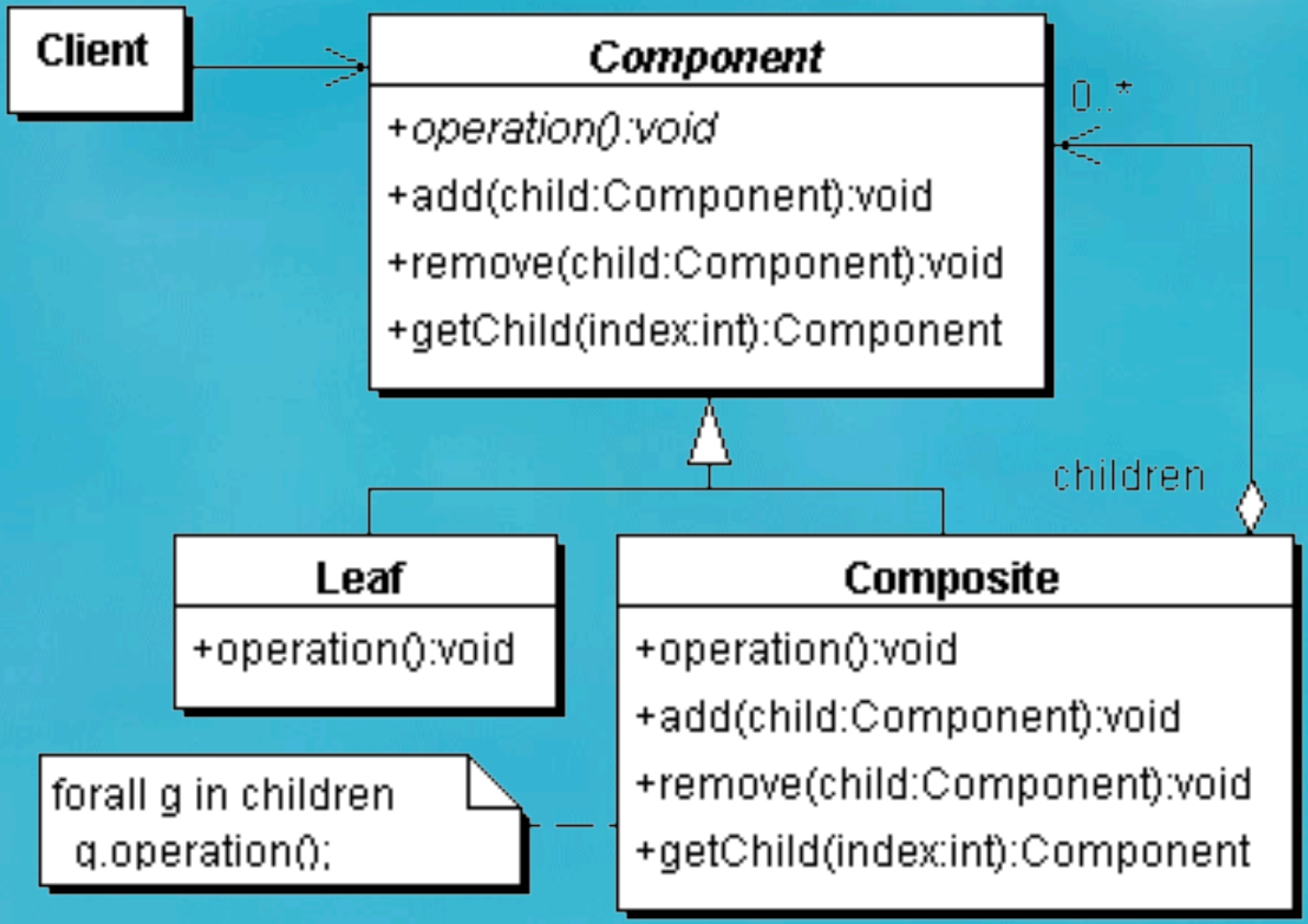
```
To: john@aol.com
Msg: welcome to the 5th edition of ...

To: amrita@intnet.mu
Msg: welcome to the 5th edition of ...

To: anton@bea.com
Msg: welcome to the 5th edition of ...
```

Applicability: Composite

- Use the Composite pattern when
  - you want to represent part-whole hierarchies of objects.
  - you want clients to be able to ignore the difference between compositions of objects and individual objects.

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Structure: Composite



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Consequences: Composite

- Benefits
  - defines class hierarchies consisting of primitive objects and composite objects
  - makes the client simple
  - makes it easier to add new kinds of components
- Drawbacks
  - can make your design overly general

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Known Uses: Composite

- java.awt.Component
- java.io.File

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Questions: Composite

- The Composite Pattern is one of the most commonly used patterns in Object Orientation. How would you go about designing the Mailing List example without this patterns, i.e. without having a common superclass?
- What maintenance issues would this cause?

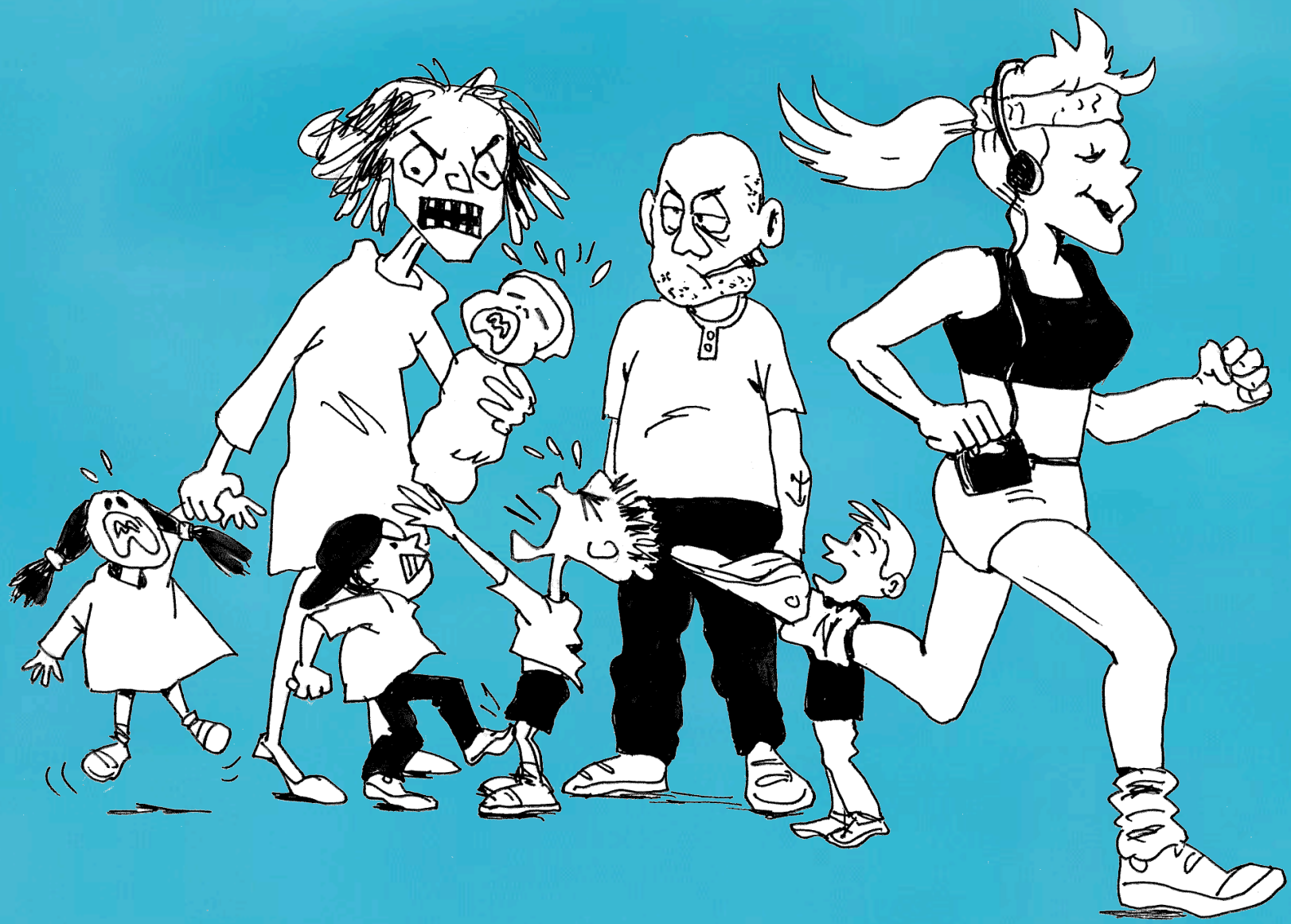
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Exercises: Composite

- Add **isLeaf():boolean** and **children():Iterator** methods to **Contact**. **children()** returns an Iterator of all children of the current contact (not recursively). Leaves would return a **NullIterator** (which is a Singleton).
- Write an external **ContactIterator** class that returns all the leaves below a **Contact**.
- Map the Contact example to a relational database.

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

3: Singleton



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

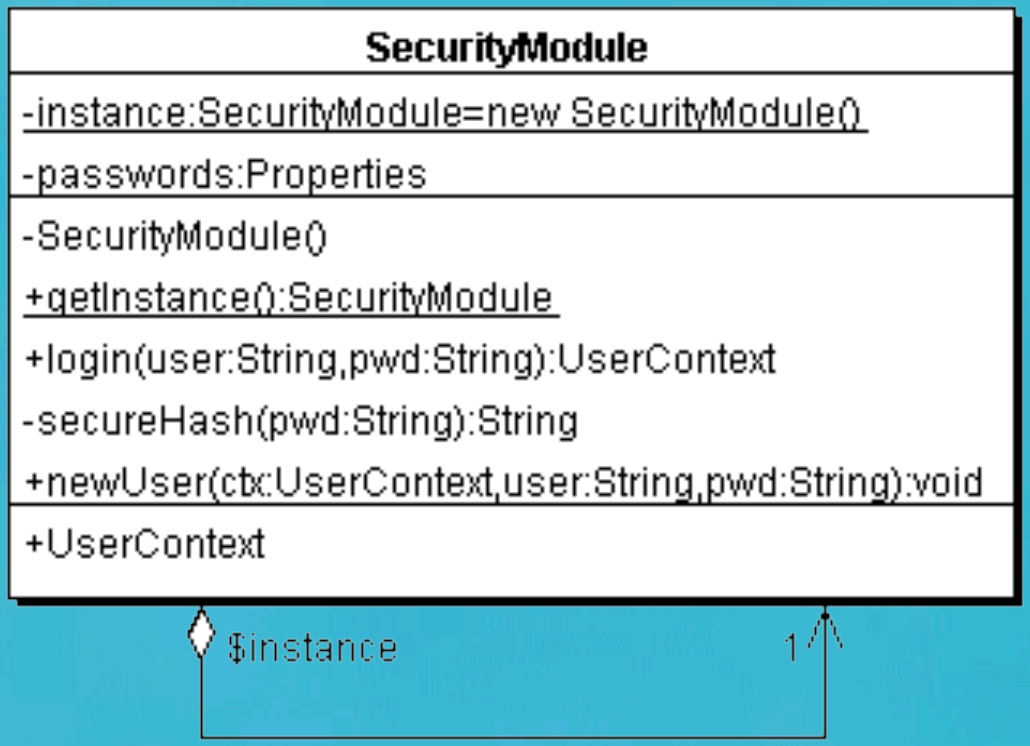
Singleton

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

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Motivation: Singleton

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



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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.
```

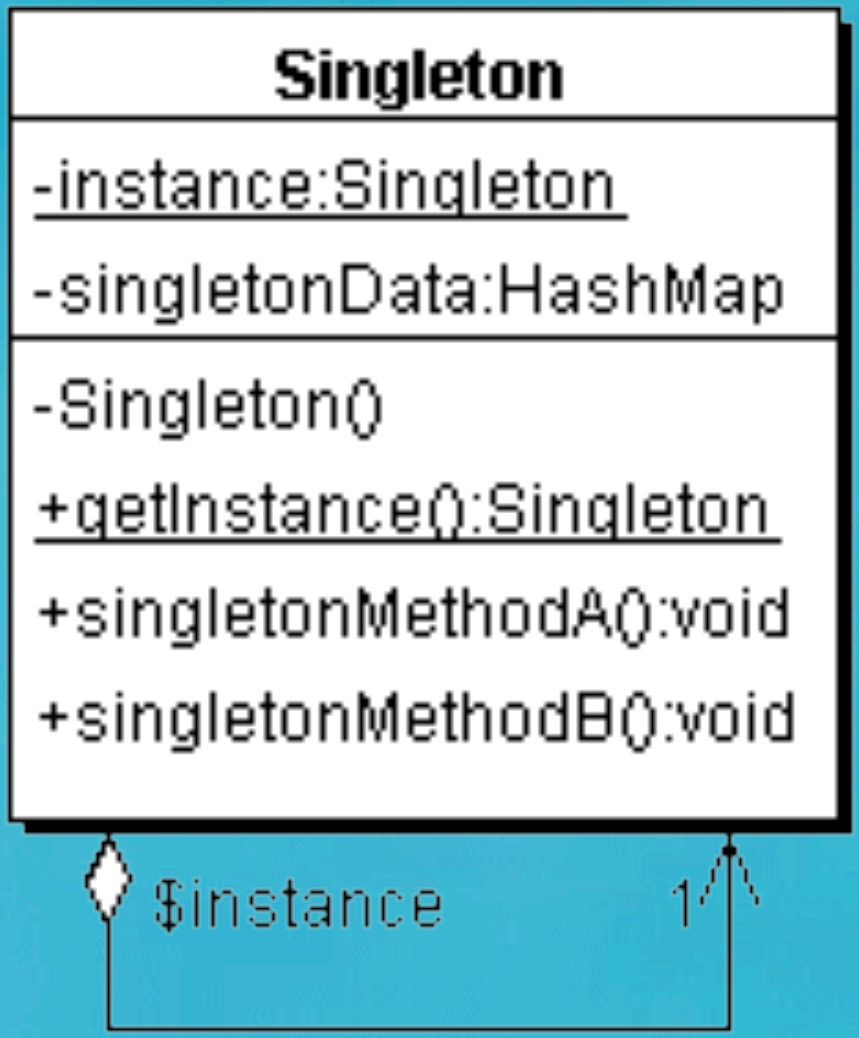
Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

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.

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Structure: Singleton



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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.

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Known Uses in Java: Singleton

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

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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?

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

4: Conclusion

- Design Patterns will help you write real Object Orientated code
- The textbook by GoF is very intimidating
- More information about how you can learn Design Patterns on:
  - http://www.javaspecialists.co.za
- Questions …
- Email: heinz@javaspecialists.co.za

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved


Design Patterns
Mauritius

Design Patterns Germany

Design Patterns South Africa

Design Patterns London


Design Patterns Estonia
at -17º Celsius