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

Dr. Heinz Kabutz

- Professional Java Programmer
- Received PhD in Computer Science from the University of Cape Town, South Africa
- Trainer of Java and Design Patterns Courses in various places of the world
- Publish advanced Java newsletter “Made in Africa” that is reaching 99 countries
  – This raises Africa’s technological image
- This is my 3rd visit to Mauritius!

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Structure of Talk

- Software Engineering
  - as it happens in the software factories
- How Design Patterns fit in
- Two examples of Design Patterns
- Discussion time

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

1. Software Engineering

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

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

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

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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!!!

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

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!

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

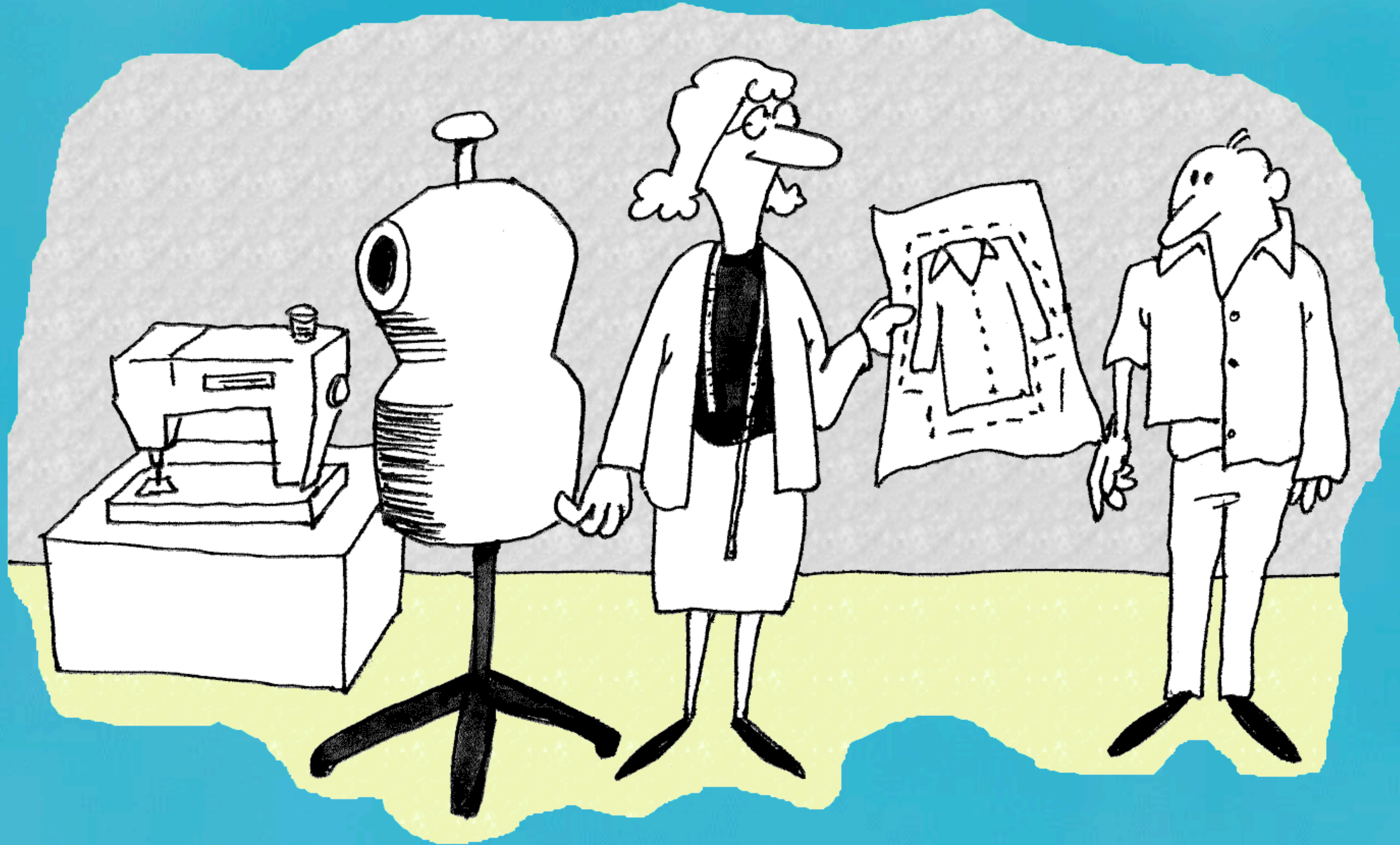
Design Language can Help

- 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

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

2: Introduction to Patterns



Vintage Whiskey

- Design Patterns are like good whiskey
  - You cannot appreciate them at first
  - As you study them you learn the difference between single-malt and normal whiskey
  - 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 cheap stuff!

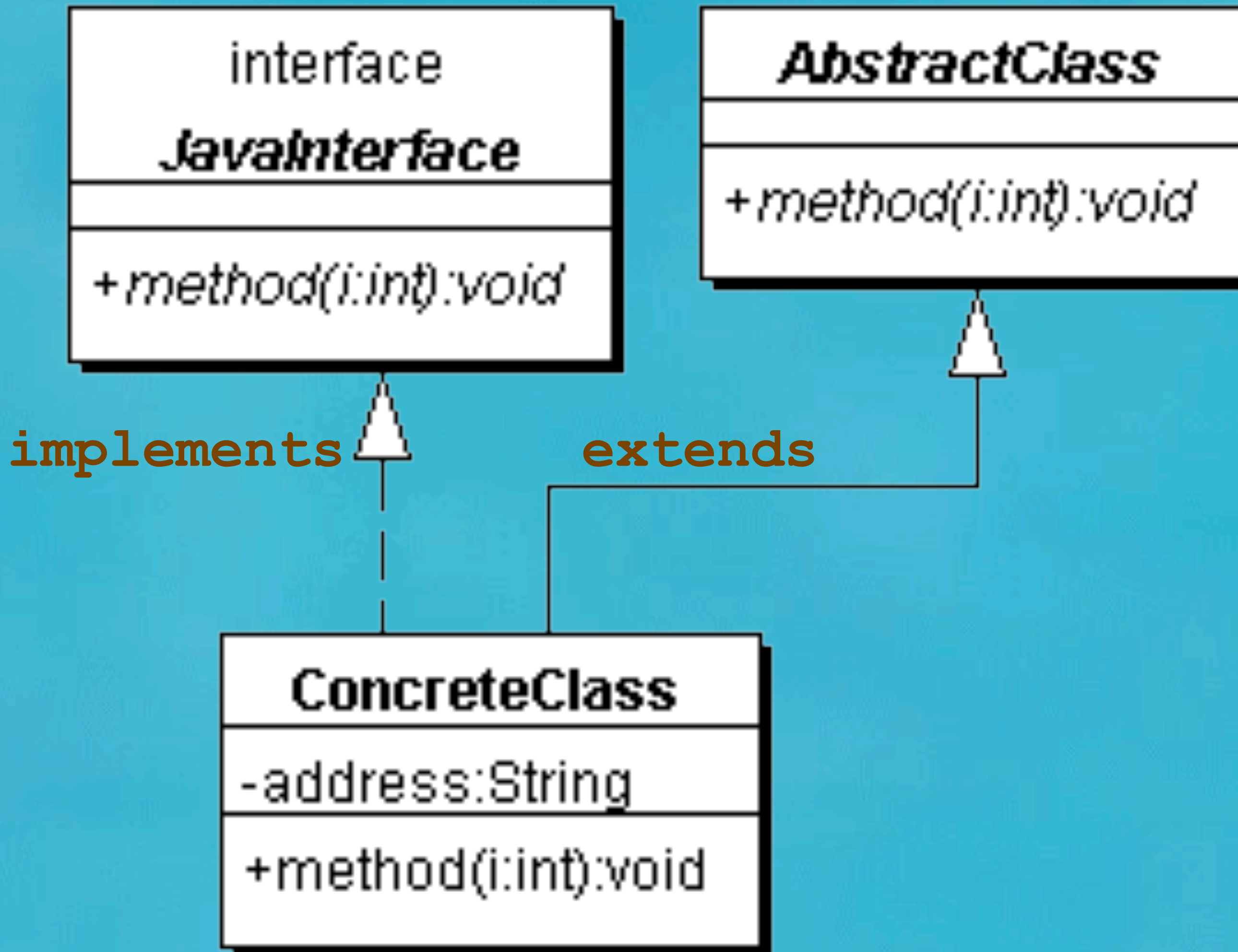
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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

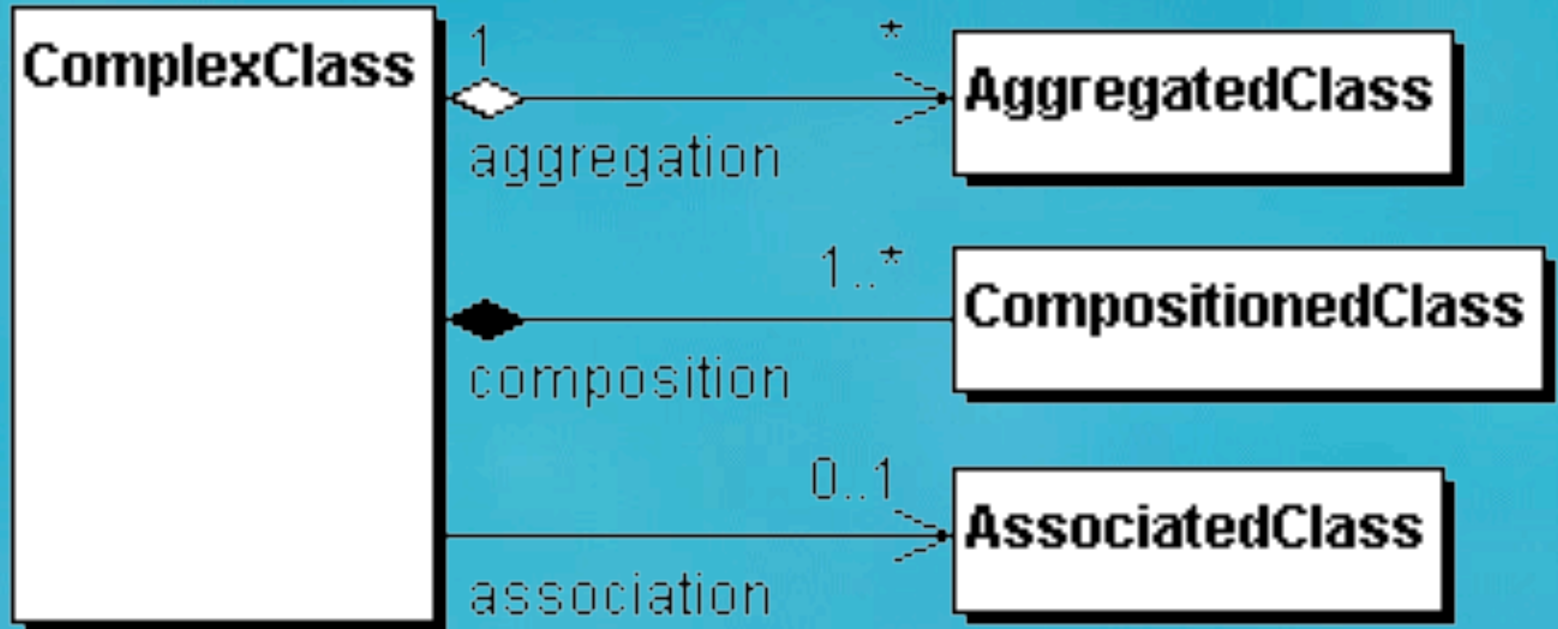
UML Refresher – Inheritance



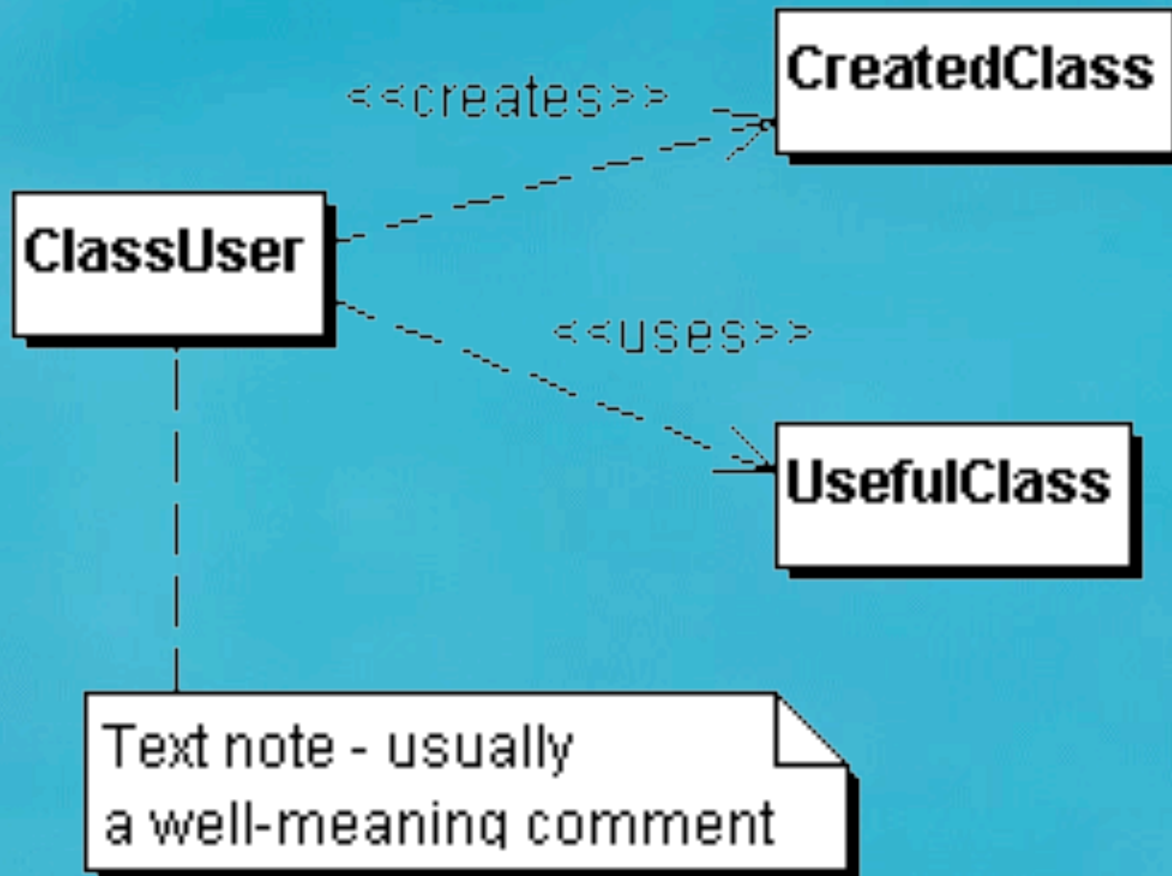
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

UML Refresher – Links



UML Refresher – Dependencies



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

UML Refresher – Access

- **public** access represented by +
- **private** access represented by –
- **protected** access represented by #
- **package** access represented by no symbol
- **static** access shown as underlined
- **abstract** methods show in Italics

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

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.

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

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”.

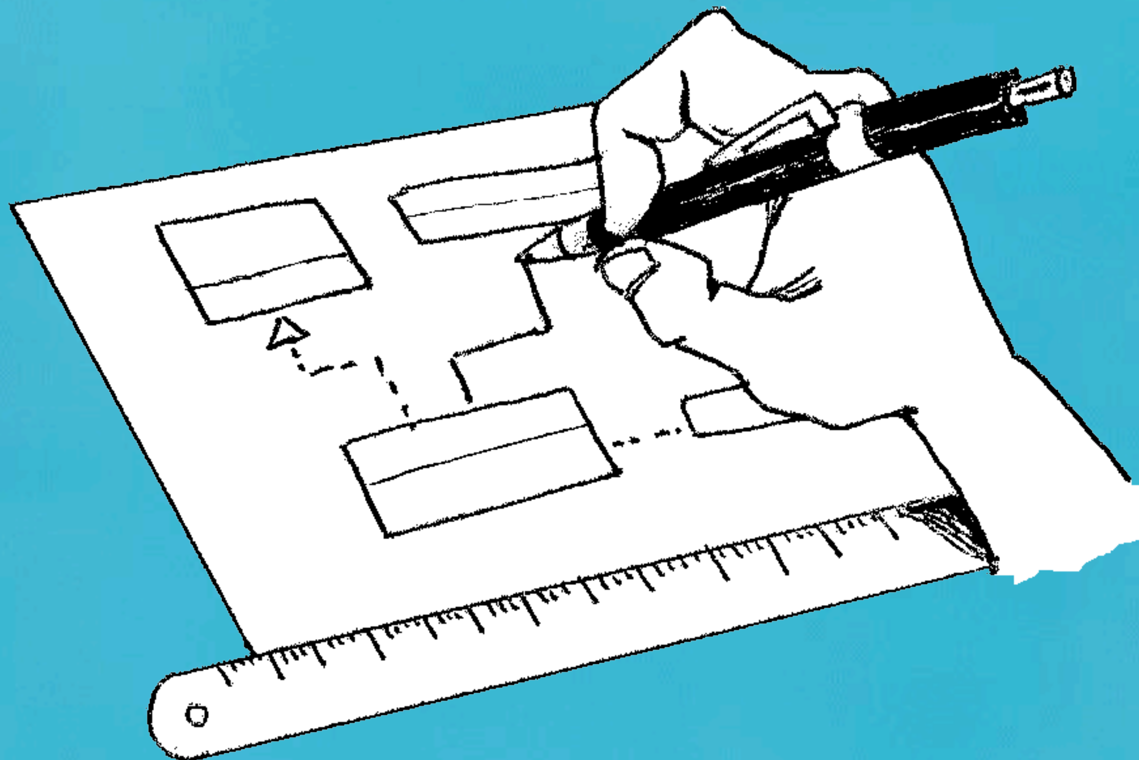
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Why do we need a diagram?

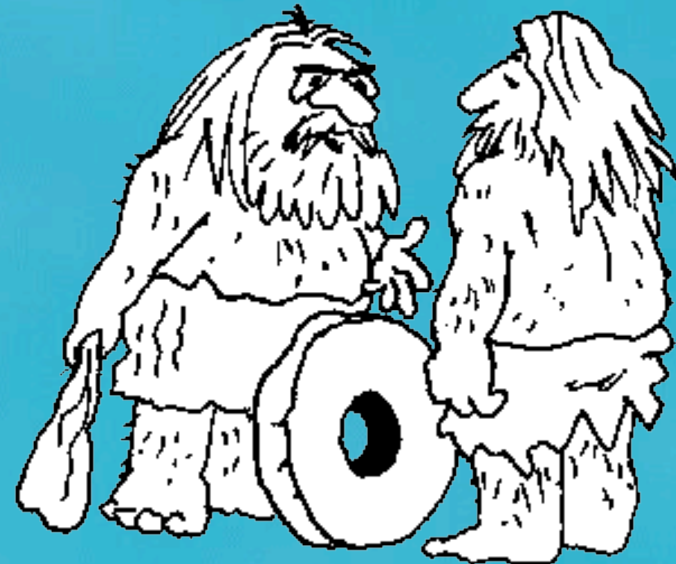
The Timeless Way of Building

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



Misuse of Design Patterns

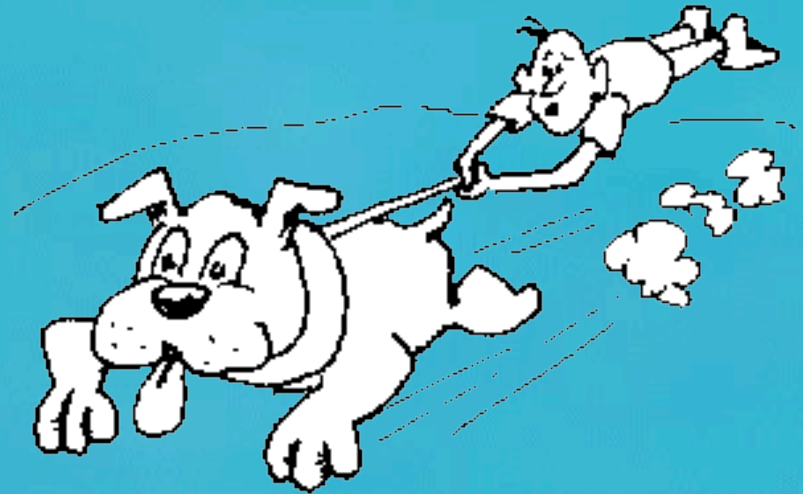
- Patterns Misapplied
  - “design” patterns should not be used during analysis
- Cookie Cutter Patterns
  - patterns are generalised solutions
- Misuse By Omission
  - reinventing a crooked wheel



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Summary

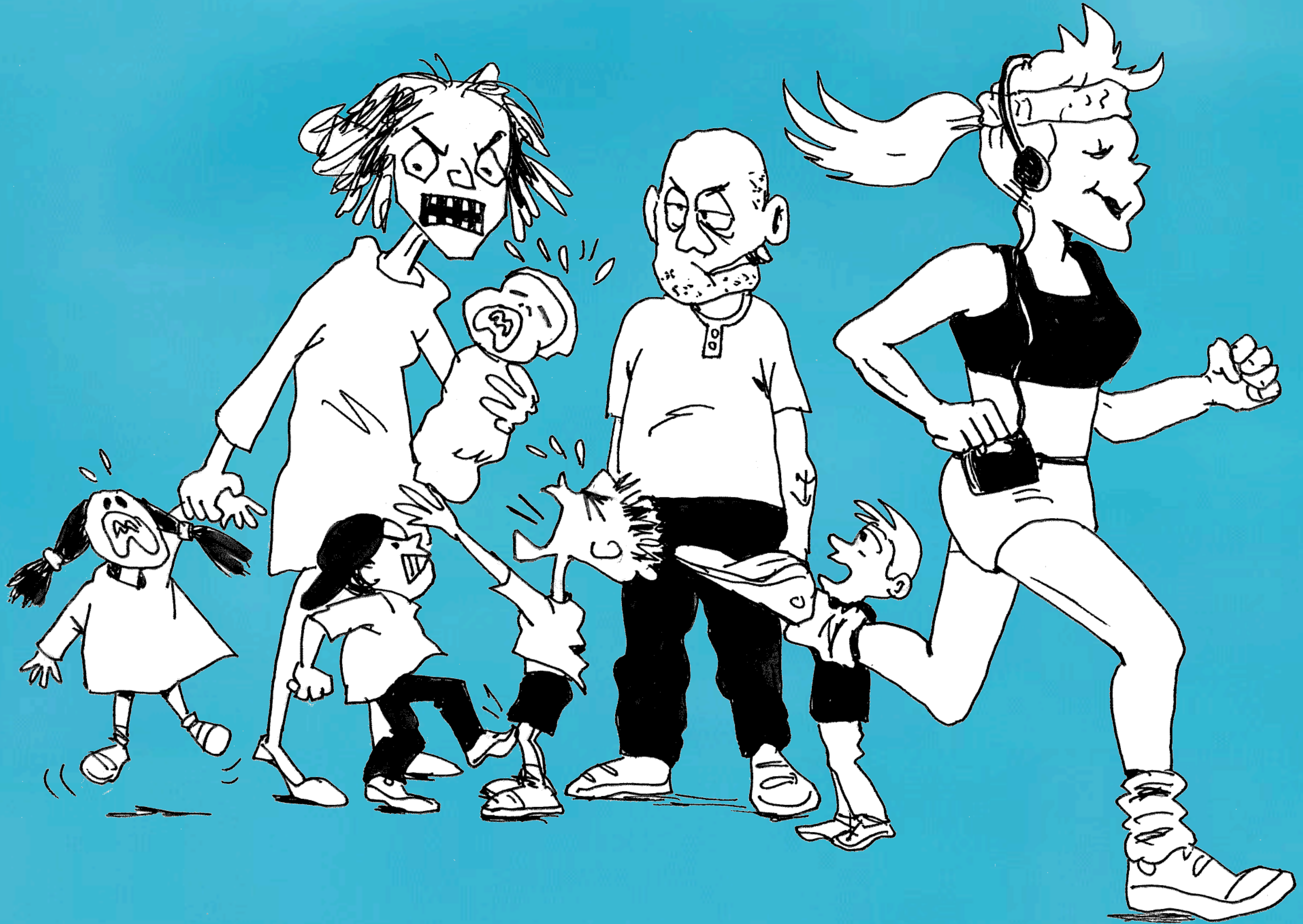
- Object Orientation is here to stay
- Design Patterns will fast-track you in learning how to design with objects



Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

3. Singleton



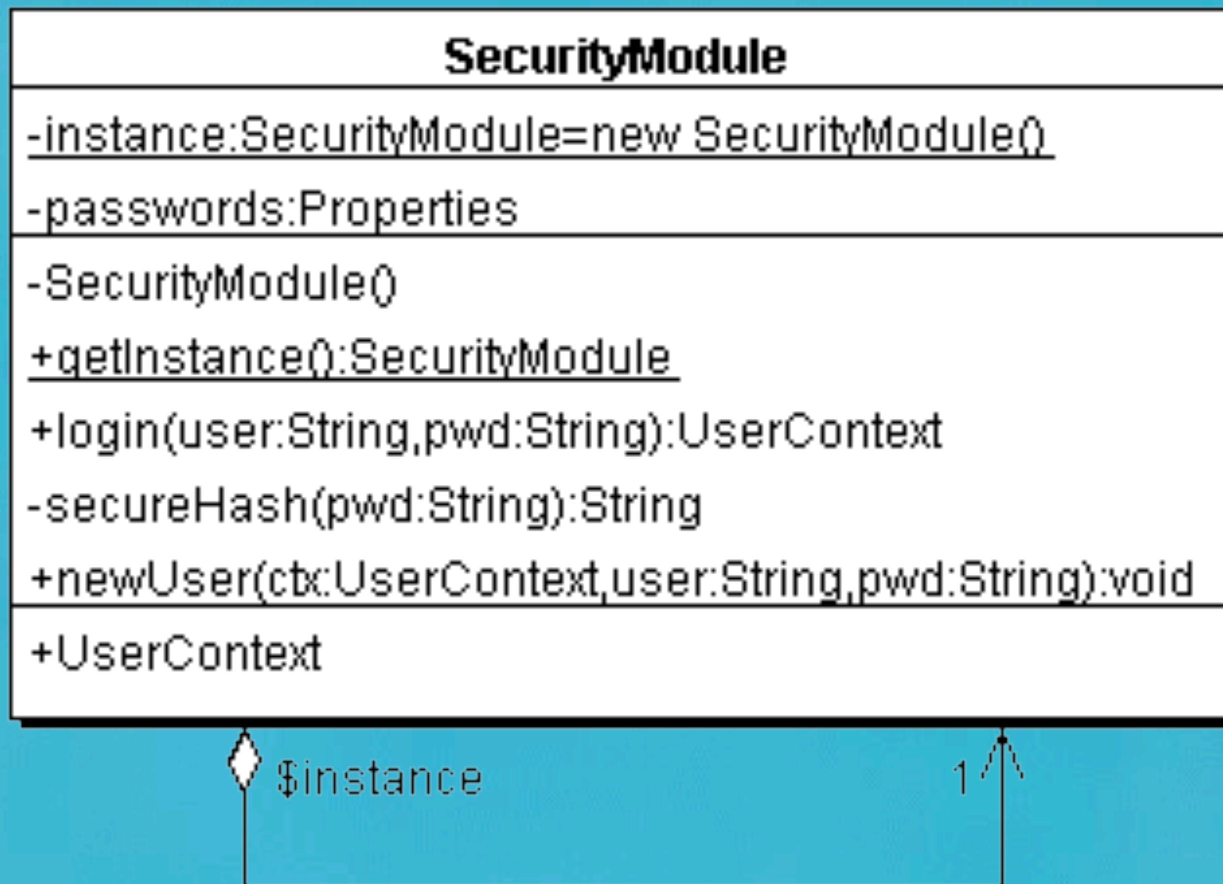
Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

Singleton

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

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.
```

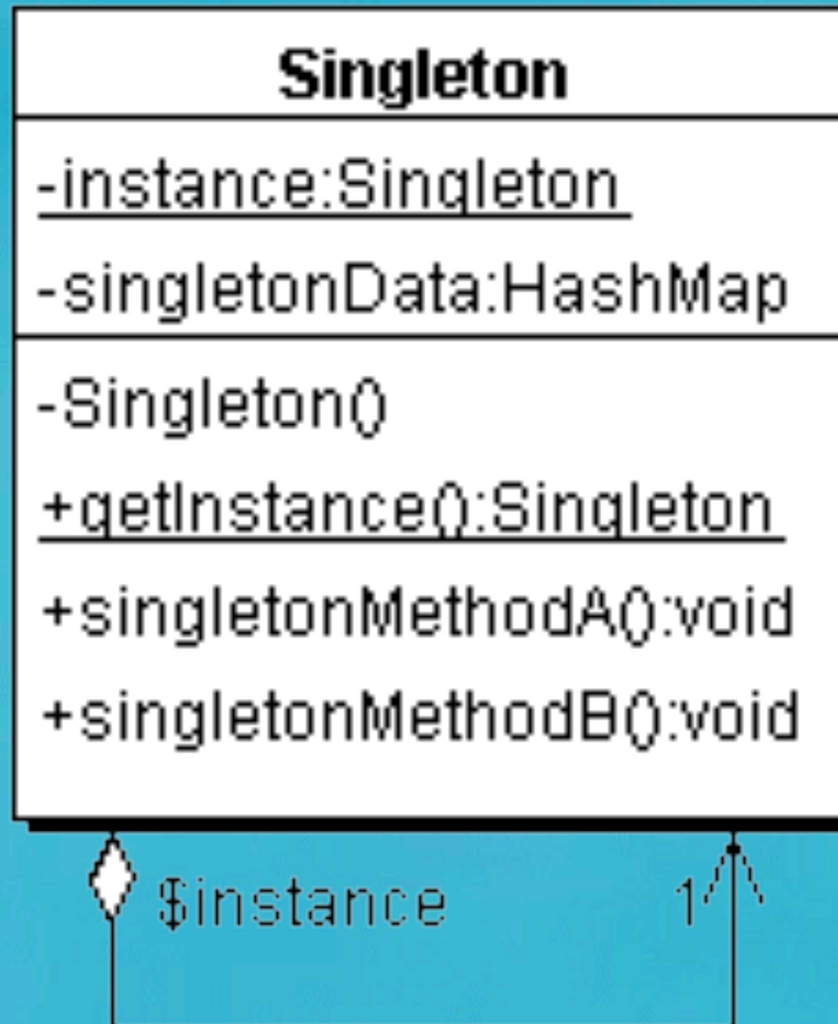
Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

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. Page-by-Page Iterator

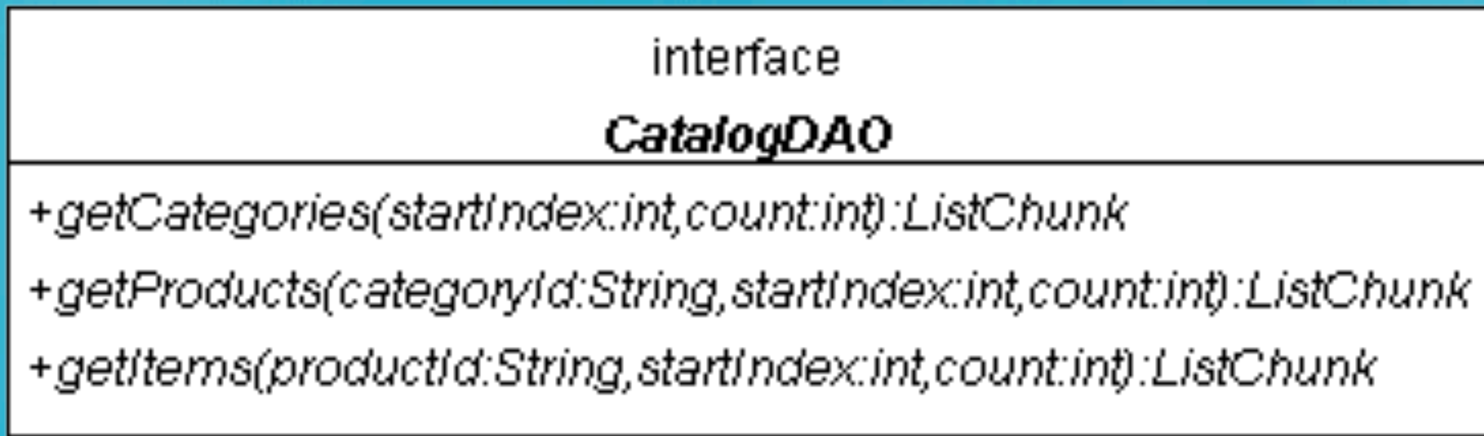
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

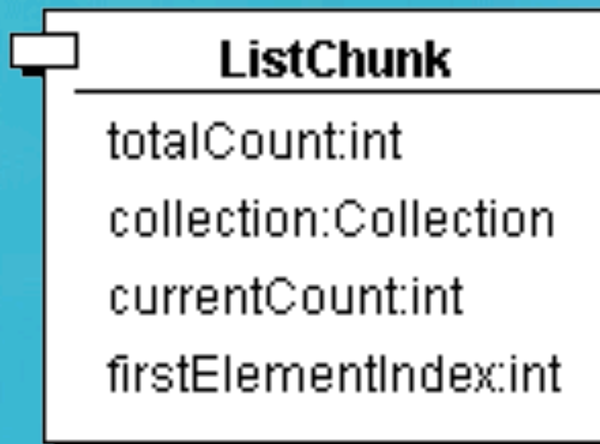
Page-by-Page Iterator

- Intent
  - Efficiently access a large, remote list by retrieving its elements one sublist of value objects at a time.
- Also known as
  - Paged List, Value List Handler

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Motivation: P-b-P Iterator





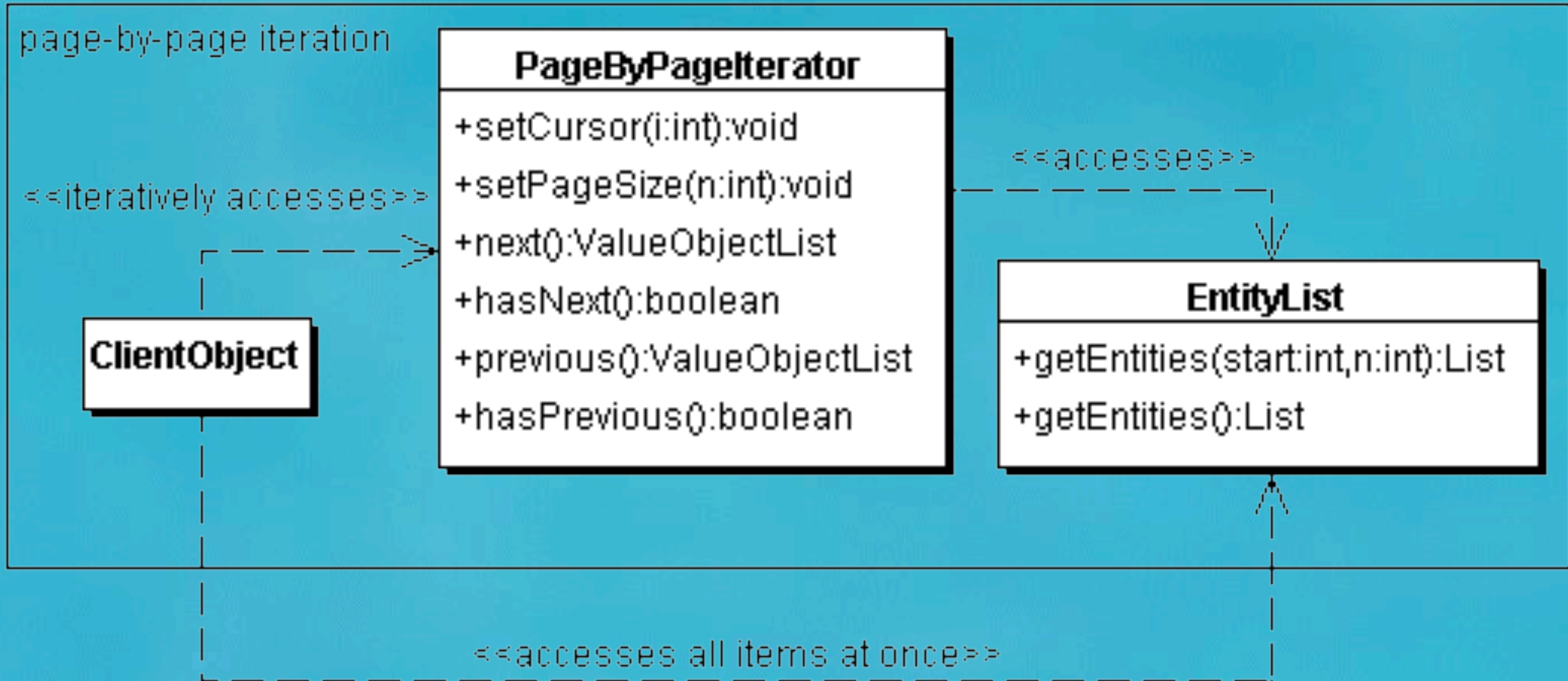
Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Applicability: P-b-P Iterator

- Use a page-by-page iterator to access a large list of server-side data when:
  - the user will be interested in only a portion of the list at any time.
  - the entire list will not fit on the client display.
  - the entire list will not fit in memory.
  - transmitting the entire list at once would take too much time.

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Structure: P-b-P Iterator



Consequences: P-b-P Iterator

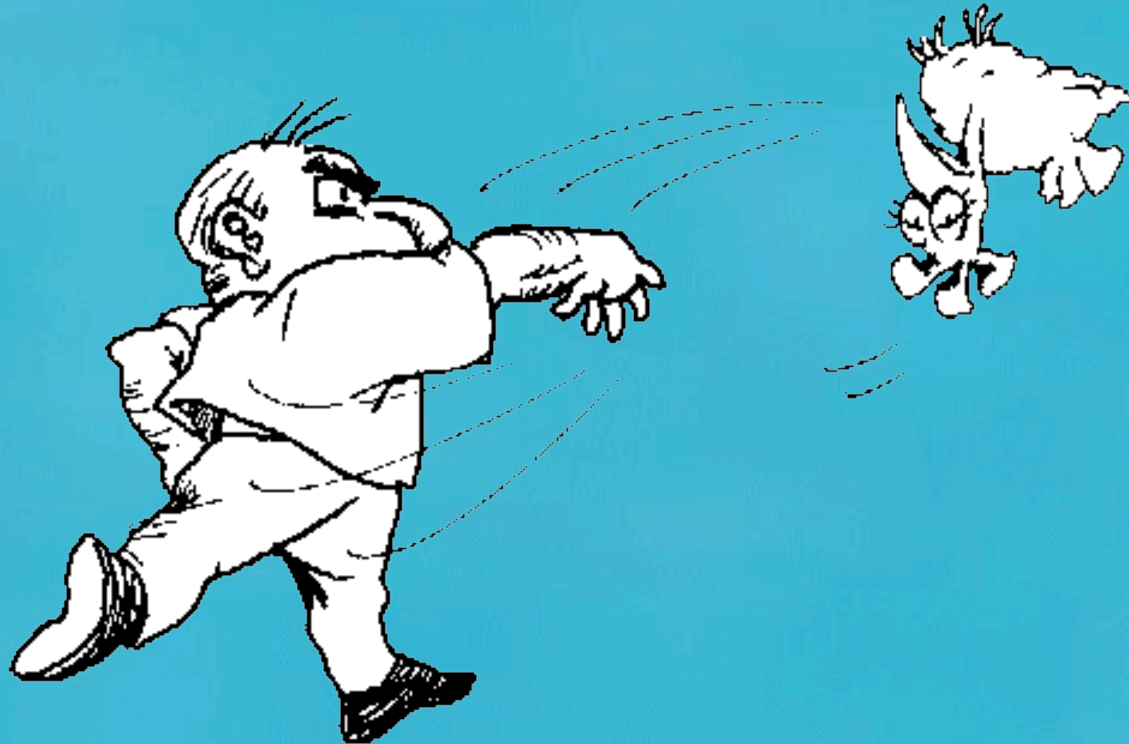
- Benefits
  - Alternative to EJB Finders for large queries
  - Caches query result on server side
  - Provides better querying flexibility
  - Improves network performance
    - Less server-side data is transferred
  - Can defer entity bean transactions
- Drawbacks
  - More server requests are made
  - The iterator is not robust

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Copyright © 2001 Maxkab Solutions CC - All Rights Reserved

Known Uses: P-b-P Iterator

- PetStore example:
  - CatalogDAO returns a ListChunk object



Questions: P-b-P Iterator

- How many rows would you need in the result set for this pattern to be useful? Why?
- What optimizations could you add to increase the speed of data retrieval?

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

Exercises: P-b-P Iterator

- Design a Page-by-Page Iterator that uses a background thread to prefetch data.
- Draw a sequence diagram of what method calls are required to fetch some data from the P-b-P Iterator.

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

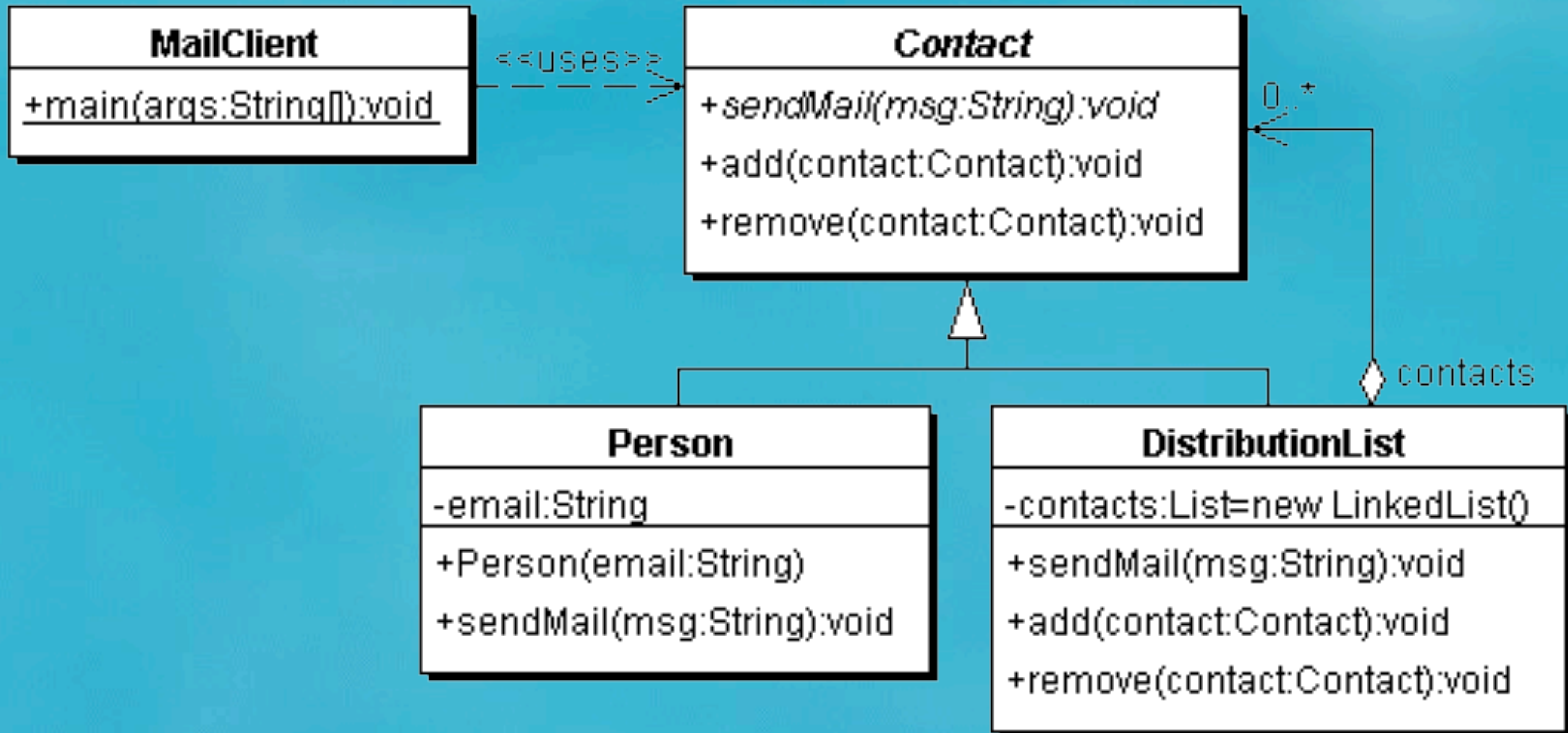
5: Composite

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("peter@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 ↵

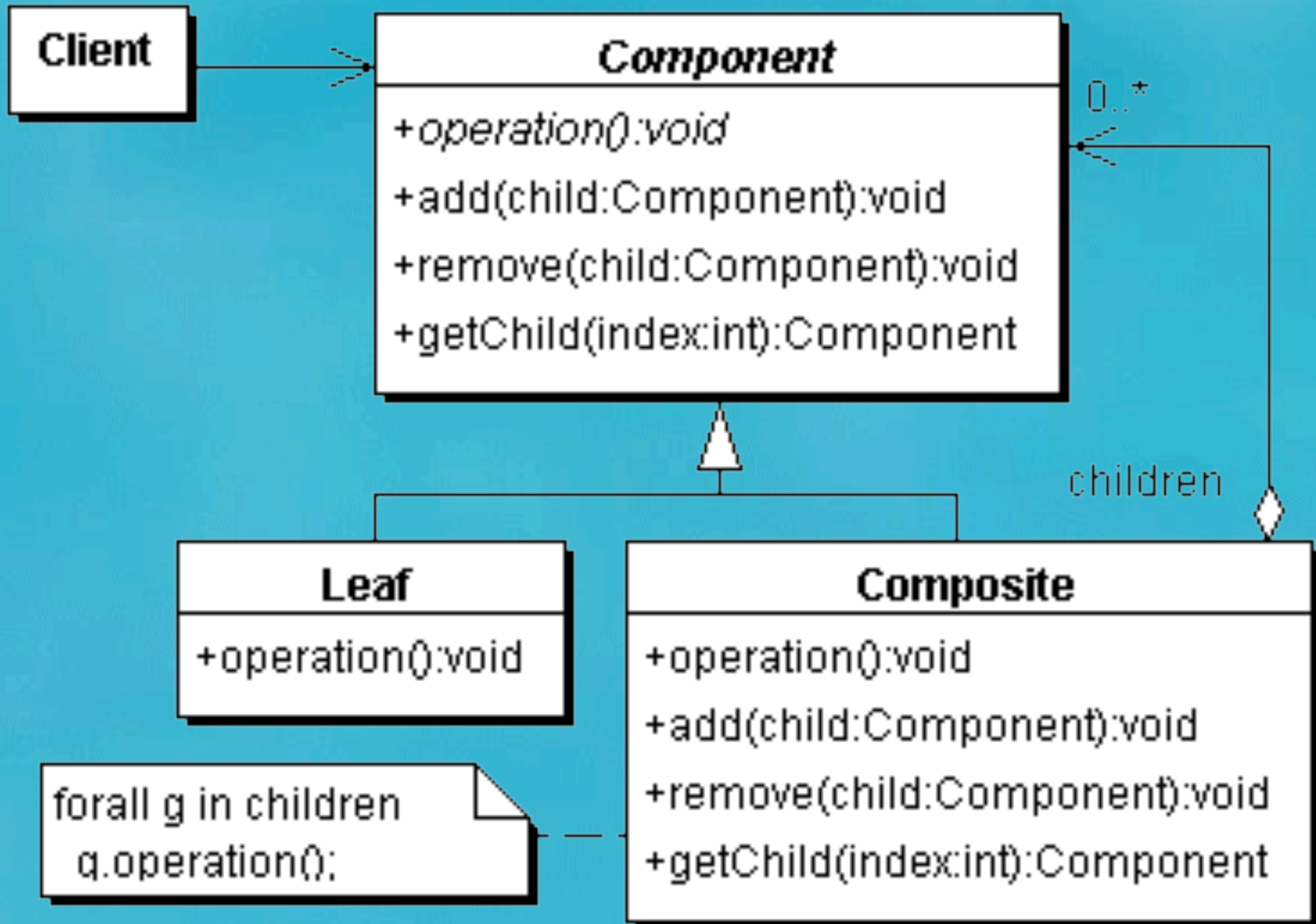
```
To: john@aol.com
Msg: welcome to the 5th edition of ...

To: peter@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

6. Design Patterns Course

- Easiest way to learn Design Patterns is through a course:
  - http://www.javaspecialists.co.za
- 3 days of action packed learning fun

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved

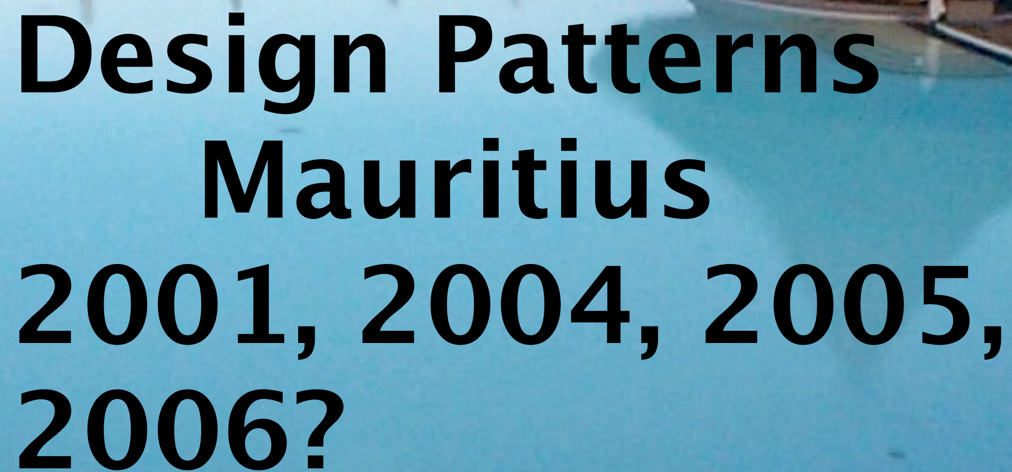
Design Patterns Cape Town

Design Patterns Germany

Design Patterns London

Design Patterns
Switzerland

Design Patterns Estonia
at -18° Celsius



Design Patterns Mauritius

2001, 2004, 2005, 2006?

My Dream

- Africa taking a technological lead
  - e.g. Mark Shuttleworth
- Mauritius as cyber island with **excellent** programmers
  - Not just cheap, but good solid quality
  - Able to compete with Eastern Europe
- Coming back to your beautiful island, year after year ☺

Copyright © 2001 Maxkab Solutions CC – All Rights Reserved