Überblick

- Von C nach Java
- Background
- Konzepte des objektorientierten Programmierens
- Ein ausgearbeitetes Java-Programm als Beispiel
- Ein weiteres Java-Programm als Beispiel
- Typen und Subtypen
- Vom Design zum Programmieren (OOD zu OOP)
- Patterns in OOP
- Testen (von objektorientierten Programmen)

Patterns in OOP

- Patterns
- Java-Design-Patterns
- Das Singleton-Pattern
- Das Observer-Pattern

Patterns

- Verallgemeinerte Lösungsansätze für wiederkehrende Probleme
- Stellen wiederkehrende Probleme mit Lösungen in Beziehung, in jeweils gegebenem Kontext
- Strukturierte Beschreibungen, die Text, Modelle (und sogar Hypermedia) verwenden
- Verwendet für Gebäudearchitektur, Softwaredesign, Analyse, Geschäftsprozesse und User Interfaces
- Pattern-Sprachen

Java-Design-Patterns

- J. W. Cooper, *Java Design Patterns: A Tutorial*, Addison-Wesley, 2000.
- Übliche Design-Patterns, implementiert in Java

Das Singleton-Pattern

- Name: *Singleton*
- Kontext: Kreieren von Instanzen
- Problem: Zu gewährleisten, dass es eine – und nur eine – Instanz einer Klasse gibt
- Lösung: Siehe Java-Code unten
- Bekannte Applikationen: Window Manager, Spooler

Das Singleton Pattern – Java-Code

```
public class ClassName {
    private static ClassName ref; //class variable
    private ClassName() {    //privatized constructor
     }
    //return only one instance
    public static synchronized ClassName getInstance() {
            if (ref == null)  //if none created yet
                    ref = new ClassName();
            return ref;
    }
    ...
}
```

Das Singleton-Pattern – Tricks

- Verwendung einer Klassenvariable für die Zustandsspeicherung, da für die Klasse gemerkt werden muss, ob es schon eine Instanz gibt
- Kreieren einer Instanz innerhalb einer statischen Methode der Klasse
- Kreieren nur dann, wenn noch keine Instanz kreiert wurde
- Deklaration des Constructors als private, um zu verhindern, dass die Klasse von außen und damit potentiell mehr als einmal instanziiert wird

Das Singleton-Pattern – Spooler-Beispiel

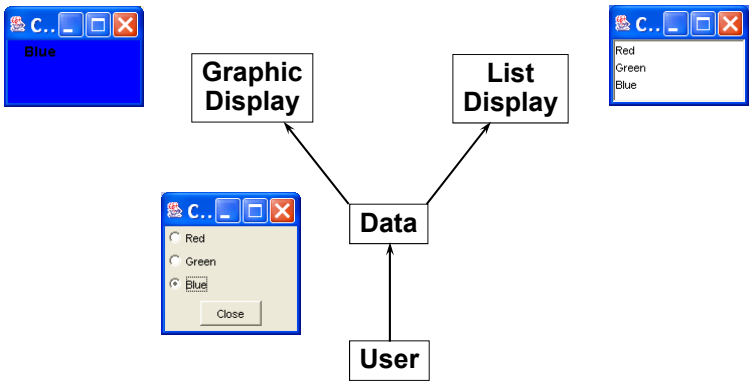
```
public class PrintSpooler {
    private static PrintSpooler spooler; //class variable
    private PrintSpooler() {    //privatized constructor
     }
    //return only one spooler instance
    public static synchronized PrintSpooler getSpooler() {
            if (spooler == null)  //if none created yet
                    spooler = new PrintSpooler();
            return spooler;
    }
    …
}
```

Das Observer-Pattern

- Name: *Observer*
- Kontext: Zustandswechsel oder Events, die für andere Objekte relevant sind
- Problem: Datenänderung an einer Stelle soll bekannt gemacht werden.
- Lösung: Ein Publisher registriert dynamisch einen oder mehrere Subscriber, die an einem Ereignis interessiert sind, und benachrichtigt sie, wenn ein Ereignis auftritt.
- Bekannte Applikationen: Verwendet für Synchronisation
- Alias: *Publisher-Subscriber*

Das Observer-Pattern – Beispiel

- Beobachten eines Farbwechsels am Bildschirm



Das Observer-Pattern – Java-Code

```
public interface Observer {    //Subscriber, e.g., display
    //notify observers that a change has taken place
    public void sendNotify(String s);
}
public interface Subject {    //Publisher, e.g., data
    //tell the subject that you are interested in changes
    public void registerInterest(Observer obs);

    //extension to Cooper book:
    //tell the subject that you are no longer interested
    //in changes
    public void unregisterInterest(Observer obs);
}
```