Communication of items

Associations, links, desing patterns

Functionality of classes

Objects are **behaviour templates**.

Classes generally:

- are stateless
- can perform stateless operations
- can be used for consultation
  calling a static method to help executing a behavior

Functionality of objects

Objects are **instance**s of classes.

Objects generally:

- have states
- can change their states
- can perform stateful operations
- can be used for consultation, composition, delegation

Communication of program elements

To use functions of other components (classes or objects), a reference is required to that component.

- For classes this is the **class name - association**
  does not change, can be used in stateless operations
- For object it is an actual **reference - link**
  could change, can be used for stateful operations

- Classes are generally stateless, do not store state informations, but can contain assiciations
- Object can store stati information, can store associations and links

How can these links be set?

Required to set a class association:

- Class is well known, can be set at desing time, when code is written

Required to set an object reference:

- An object is required for a link to point (refer to) – target
- An object is required which will point to the referenced object – source

Link settings

- Exactly one: 1
- Zero or one: 0..1
- Zero or more: 0..*
- One or more: 1..*
- Interval: 3..8
- Discrete values: 2, 4, 6, 8

Link **data member**	Ctor, get, set
Link **data member** with NULL value	No restriction
ArrayList, can be empty	No restriction
ArrayList, can NOT be empty	Ctor, add, remove
ArrayList with controlled number of items	Ctor, add, remove
ArrayList with controlled number of items	Ctor, add, remove

Who can set the link?

Required:

- A source object (link from)
- A target object (link to)

A link can be set by who knows both!

For the first link, this is the application entry point:

```
Main.main(String[] args)
```

When references can be set?

Class associations:

- Desing time (creating the program)

Object links:

- Runtime – any time, when link is needed

Usage of references

- Design patterns are predefined and accepted structures
- To achieve a certain optimization aim
- Multiple patterns can be used parallel (combine them)
- There is no ground truth for combination (depending on optimization)

Delegation

- In object-oriented programming, **delegation** refers to evaluating a member (calling a method) of one object (the receiver) in the context of another original object (the sender).
- more precisely referred to as request *forwarding, when receiver has no information about the (context of the) sender*

Singleton

Singleton pattern should be used when we must ensure that only one instance of a class is created and this instance must be available through all the code.

A special care should be taken in multi-threading environments when multiple threads must access the same resources through the same singleton object.

Factory

- Factory method – abstract method for object creation
- Object Factory – creation of object implementing an interface
- Abstract factory – creation of descendant of an abstract class

Factory method

The factory method is a part of the class using the created object.

This class defines an abstract method for creating object, but let subclasses decide which class to instantiate and refers the newly created object through a common interface.

Created object has to implement the referred interface.

Object/Interface Factory

Used when a framework delegate the creation of objects derived from a common superclass to the factory.

With interfaces this is a tool for decoupling the code.

- Creates objects without exposing the instantiation logic to the client
- Refers to the newly created object through a common interface.

Abstract factory

A factory for creating a family of related objects, without explicitly specifying their classes.

Family means a subbranch of the abstraction hierarchy. A family of related objects is defined by their parent. This parent is usually abstract.

The facory works with children of an abstract base, not interface implementations.

Command

An object is used to encapsulate all information needed to perform an action or trigger an event at a later time.

More information is not needed and another object, can execute the action.

Allows the parameterization of clients with different requests and Allows saving the requests in a queue.

Observer

Define a one-to-many link between objects so that when one object changes its state, all linked objects (dependents) are notified and updated automatically.

This is typically done by calling one of their methods.

The pattern consists of two actors, *the observer* who is interested in the updates and *the subject* who generates the update events.

Observer

A subject can have many observers: a one to many

An observer is free to subscribe to updates from other subjects: one to many

Two one to many relations form a many-to-many relation

This pattern can:

- change notification type from pull to push
- reduce storage size, when there are a few subject only

Event handling

Implementation of the observer model

Event handling

- Changing the state of an object (the source) is called an event
- Listeners are listening for events
- On occurrence of an event, an event handler method is called
- Its parameter is the event object, containing event properties

Events vs Observers

Similarities:

- Changing the state of an object (the source) is called an event
- Listeners are listening for events (like observers)
- Observers can observe for ChangeEvent method call

Events vs Observers

Differences:

- Source can call any method of the observer – this is not an event
- Event details are presented as handler parameters
- Extension of schema: new event type / new callback method
- Event dispatcher queue/thread could handle events – async sources and listeners are decoupled

Controllers

Connection of View and Model

MVC design pattern

Model–View–Controller (MVC) is an architectural pattern divides an application into three interconnected parts.

Designed to separate internal representations of information from the ways information is presented to and accepted from the user.

Traditionally used for desktop graphical user interfaces (GUIs), but has become popular for designing web applications.

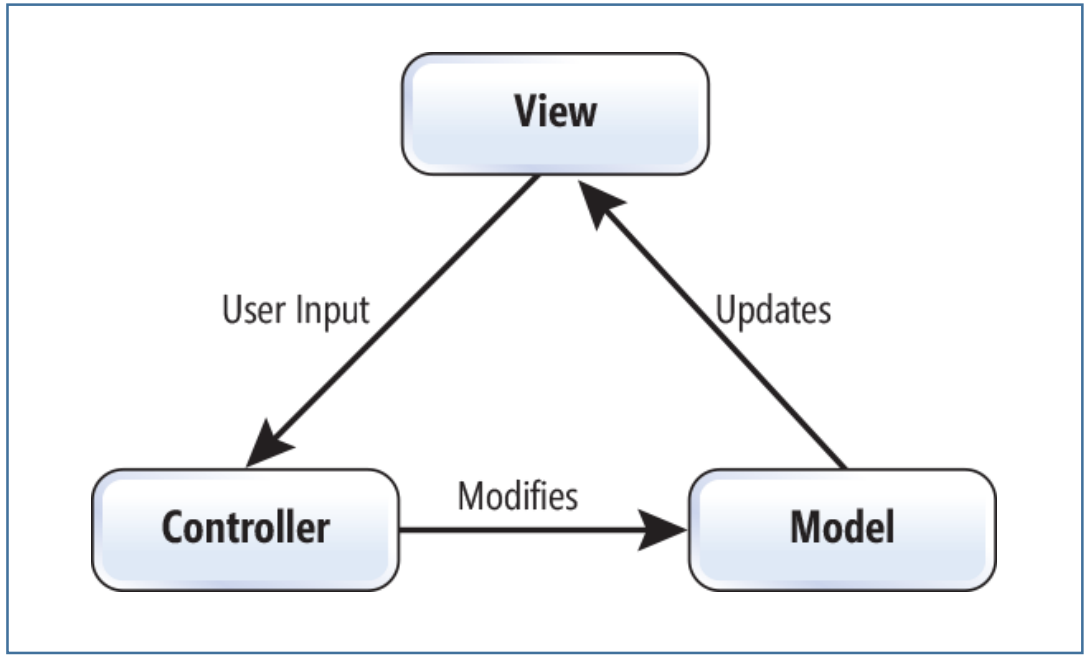
MVC parts

- **Model**
  - The central component of the pattern
  - It is the application's dynamic data structure, independent of the user interface.
  - It directly manages the data, logic and rules of the application.
- **View**
  - Any representation of information
  - Multiple views of the same information are possible.
- **Controller**
  - Accepts input and converts it to commands for the model or view.

View and Controller

In MVC model, Controller connects View and Model. Writes Model to View and handles UI events to update Model.

In a console application, View could be a combination of `System.In` and `System.Out`
When creating a Controller these are passed as parameters



Single responsibility of controller

- Model classes are responsible to store single units of data
- Assumption: View is responsible to create single representation of Model data
- Create a controller to control a single View
- **To make it easier to use System.Out as output, combine View and Controller**

Communication of controllers

- For more representations, more controllers exist
- A controller has to create/initialize/activate another

1. Create the main View and its Controller is main method
2. Create & initialize more View and Controller instances in main Controller
3. Pass the controll to subviews (show them) – modal/non-modal

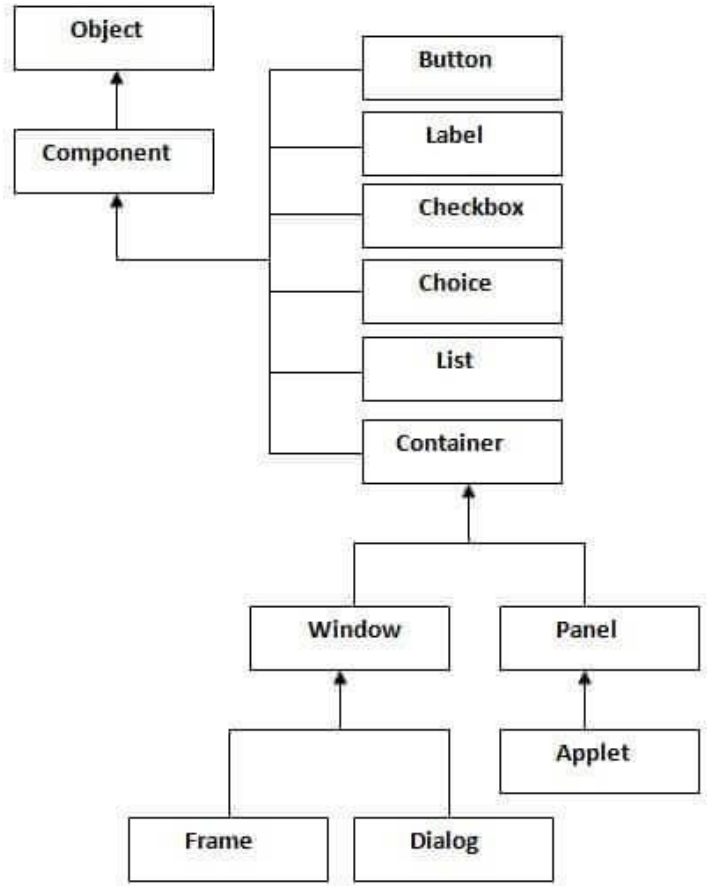
Graphical User Interface

AbstractWindowToolkit framework

Java Abstract Window Toolkit framework

- An API to develop GUI or window-based applications in java.
- Components are platform-dependent:
  components are displayed according to the view of current operating system.
- The java.awt package provides classes:
  `Button, Label, TextField, TextArea,`
  `CheckBox, CheckBoxGroup (RadioButton),`
  `Choice (DropDownList), List`

AWT class hierarchy



AWT example – create

```
class FirstFrame extends Frame {
        FirstFrame(){
                super("Java AWT example");
                Button b=new Button("click me");
                b.setBounds(30,100,80,30);
                add(b);
                setSize(300,300);
                setLayout(null);
                setVisible(true);
        }
}
```

AWT example – create

```
class FirstFrame extends Frame {
        TextField tf;
        FirstFrame(){
        tf=new TextField();
        tf.setBounds(60,50,170,20);
                …

                b.addActionListener(this)
                setVisible(true);
        }
}
```

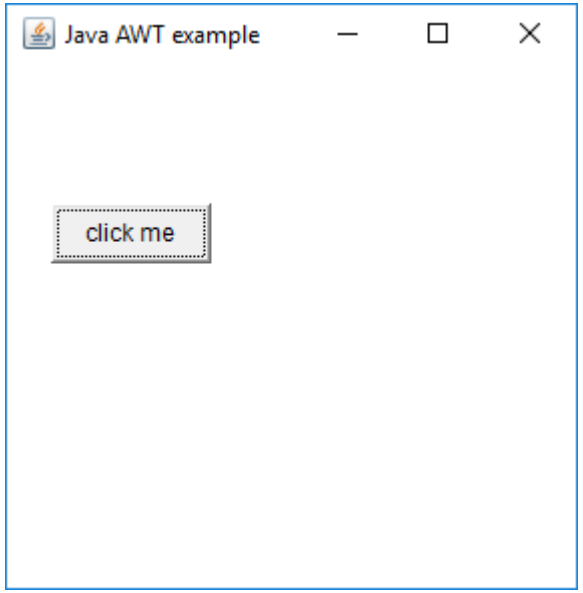
AWT example – create

```
class FirstFrame extends Frame {
        TextField tf;

    public void actionPerformed(ActionEvent e){
        tf.setText("Welcome");
    }
}
```

AWT example – show

```
class Main {
	public static void main(String args[]){
		FirstFrame f = new FirstFrame();
	}
}
```



AWT event types

- **Foreground Events** - which **do** require direct user interaction
  Generated as consequences of a user interacting with the graphical controls.
  clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page
- **Background Events** - which **do not** require directuser interaction
  operating system interrupts, hardware or software failure, timer expires, an async operation completions are examples of background events

Working with GUI

- Threading is essential for GUIs with fluent user experience, not to block the input/output thread of the user interface.
- If everything is done in the UI thread, user interctions are blocked while a long data processing operation is executed.

USERS DO NOT LIKE TO WAIT!

Java GUI frameworks

- **Java AWT** – introduced in JDK 1.0, most components are obsolete
- **Swing** – a part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC has been integrated into core Java since JDK 1.2.
- **JavaFX** – integrated to Java 8, replacing Swing

With MAVEN, framework plugins can be downloaded automatically.

JavaFX imports

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
```

JavaFX application

```
public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }
}
```

JavaFX UI

```
public void start(Stage primaryStage) {
    primaryStage.setTitle("Hello World!");
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    StackPane root = new StackPane();
    root.getChildren().add(btn);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();
}
```

JavaFX event handling

```
btn.setOnAction(new
EventHandler<ActionEvent>() {
      @Override
      public void handle(ActionEvent event) {
            System.out.println("Hello World!");
      }
});
```