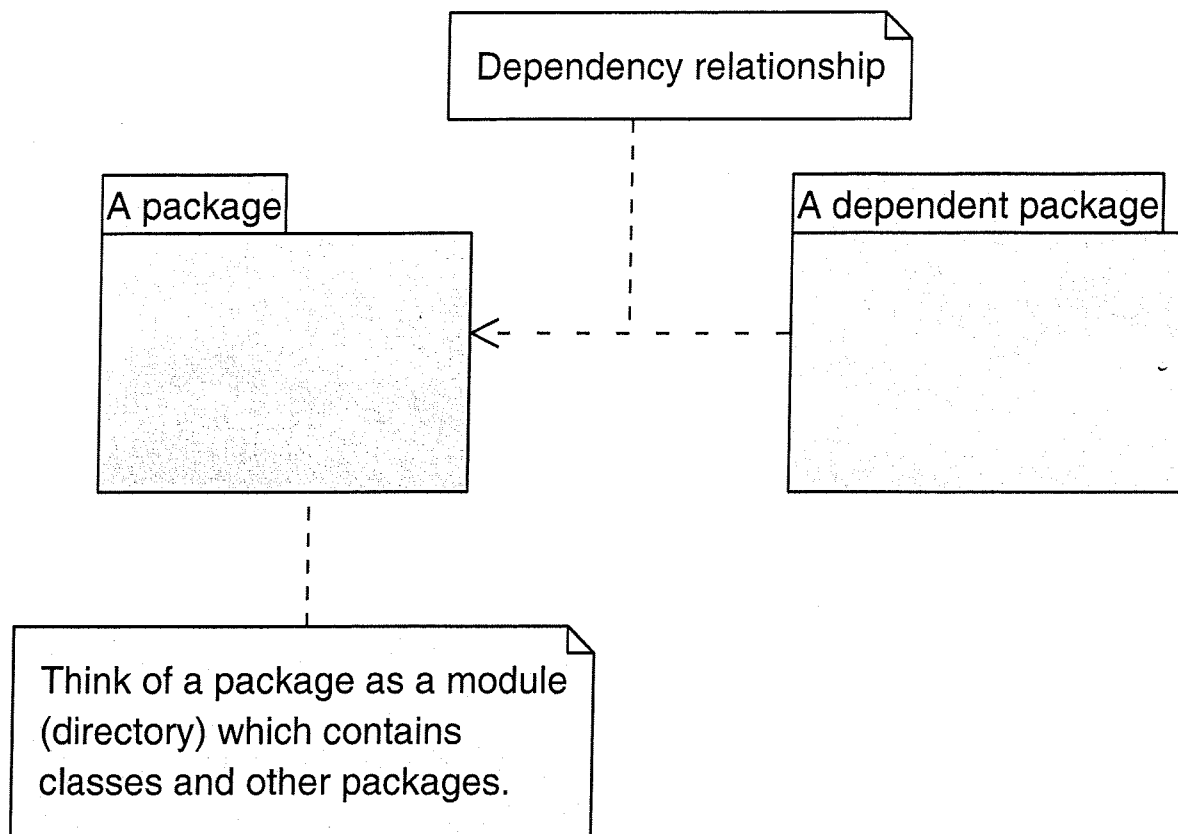


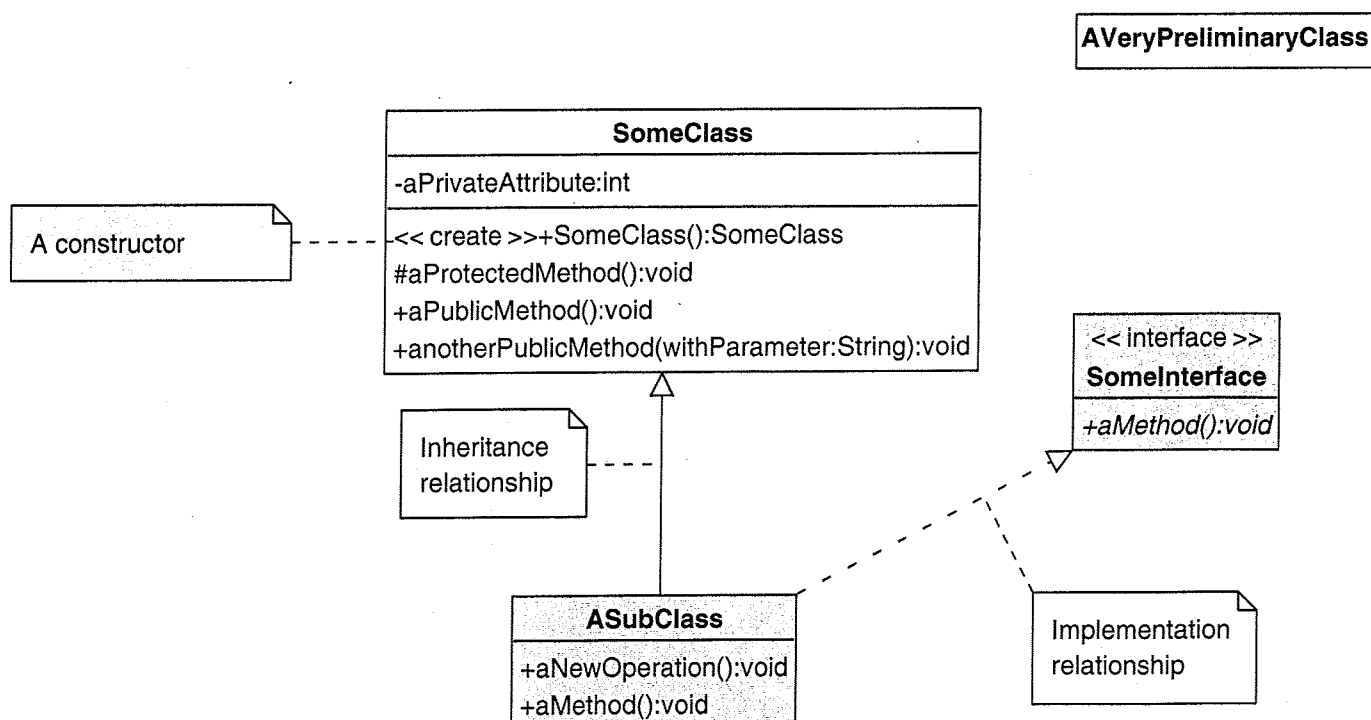
Packages

UML paket



Class icons and inheritance

UML klasser



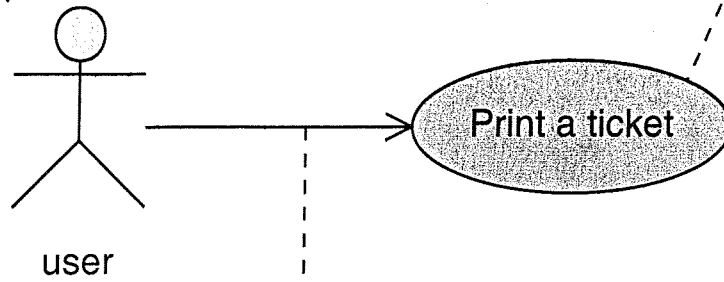
A use case diagram

UML use case

Ex. 7

An actor

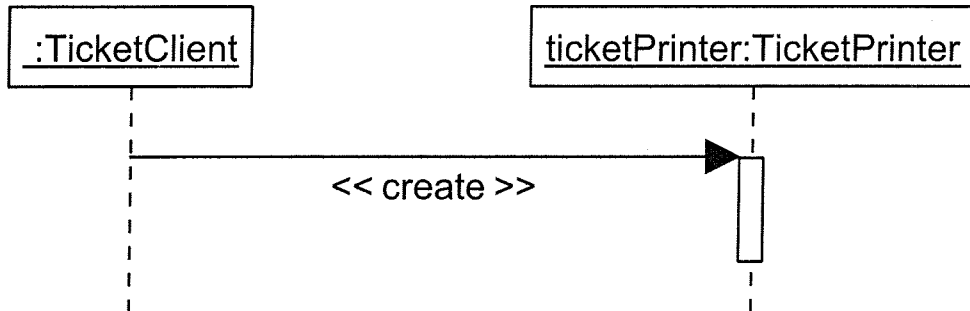
A use case



Initiation relationship

Scenario Ticket printer creation

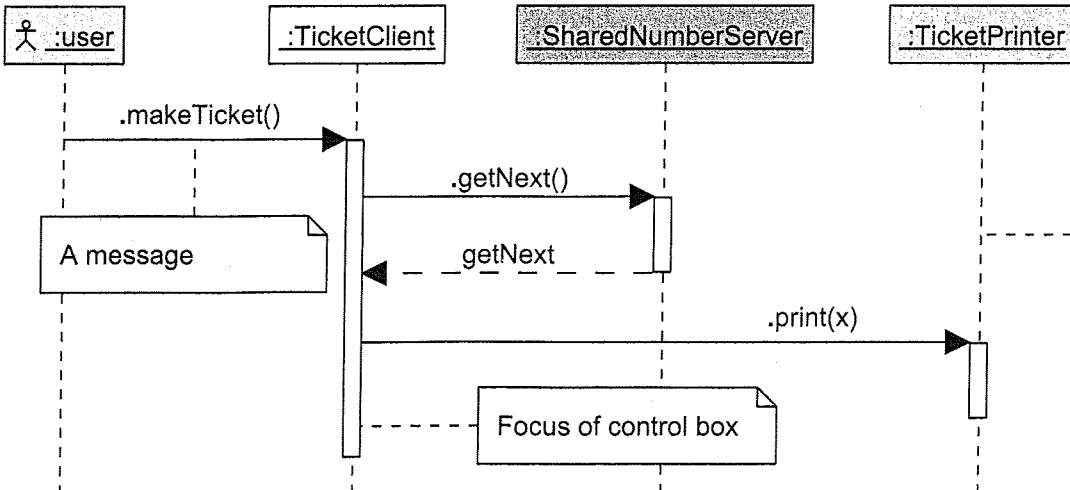
UML sekvensdiagram



Scenario: Print a ticket

A "life line" represents the life of an object over time

An actor

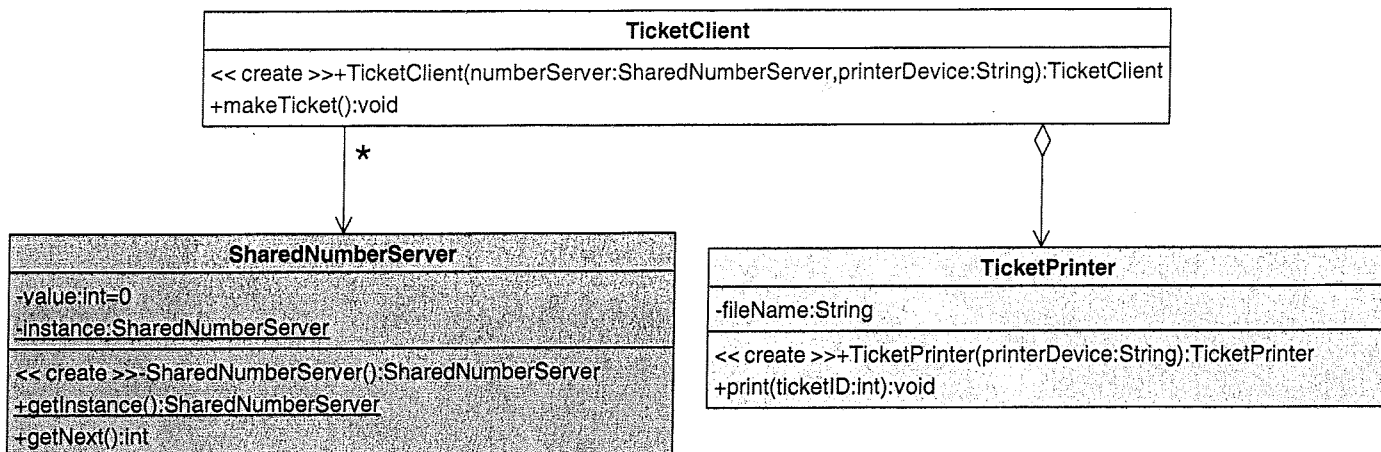


A message

Time flows downwards

Focus of control box

Aggregation and association



Class Handouts_master

1/1

```

public class TicketClient {
    private TicketPrinter ticketPrinter;
    private SharedNumberServer sharedNumberServer;

    public TicketClient(SharedNumberServer sharedNumberServer,
        String printerDevice)
    {
        this.sharedNumberServer = sharedNumberServer;
        ticketPrinter = new TicketPrinter(printerDevice);
    }

    public void makeTicket() {
        ticketPrinter.print(sharedNumberServer.getNext());
    }
}

/*****
// This class implements the Singleton design pattern
public class SharedNumberServer {
    private int value = 0;
    private static SharedNumberServer instance = null;

    private SharedNumberServer() {}

    public static SharedNumberServer getInstance() {
        if ( instance == null )
            instance = new SharedNumberServer();

        return instance;
    }

    public int getNext() {
        return value++;
    }
}
*****/

public class TicketPrinter {
    private String printerDevice;

    public TicketPrinter(String printerDevice) {
        this.printerDevice = printerDevice;
    }

    public void print(int ticketID) {
        // create a ticket with ID ticketID
        // and send it to the printer device
        // ... but here we cheat a bit, after all it's just a prototype
        System.out.println("Printer: " + printerDevice +
            ", ticket no: " + ticketID);
    }
}
  
```

Ex. 2 MVC och Observer

Use case: Ask for next prime number

1. **The use case starts here.**
2. The user pushes the button labelled "Next prime number" in the GUI.
3. The smallest prime number following the number that is displayed in the output field in the GUI is computed by the application.
4. The new number is displayed in the output field.
5. **The use case ends here.**

Use case: Reset prime number generator

1. **The use case starts here.**
2. The user pushes the prime number generator's "Reset" button in the GUI.
3. The smallest prime number is computed by the application.
4. The new number is showed in the output field.
5. **The use case ends here.**

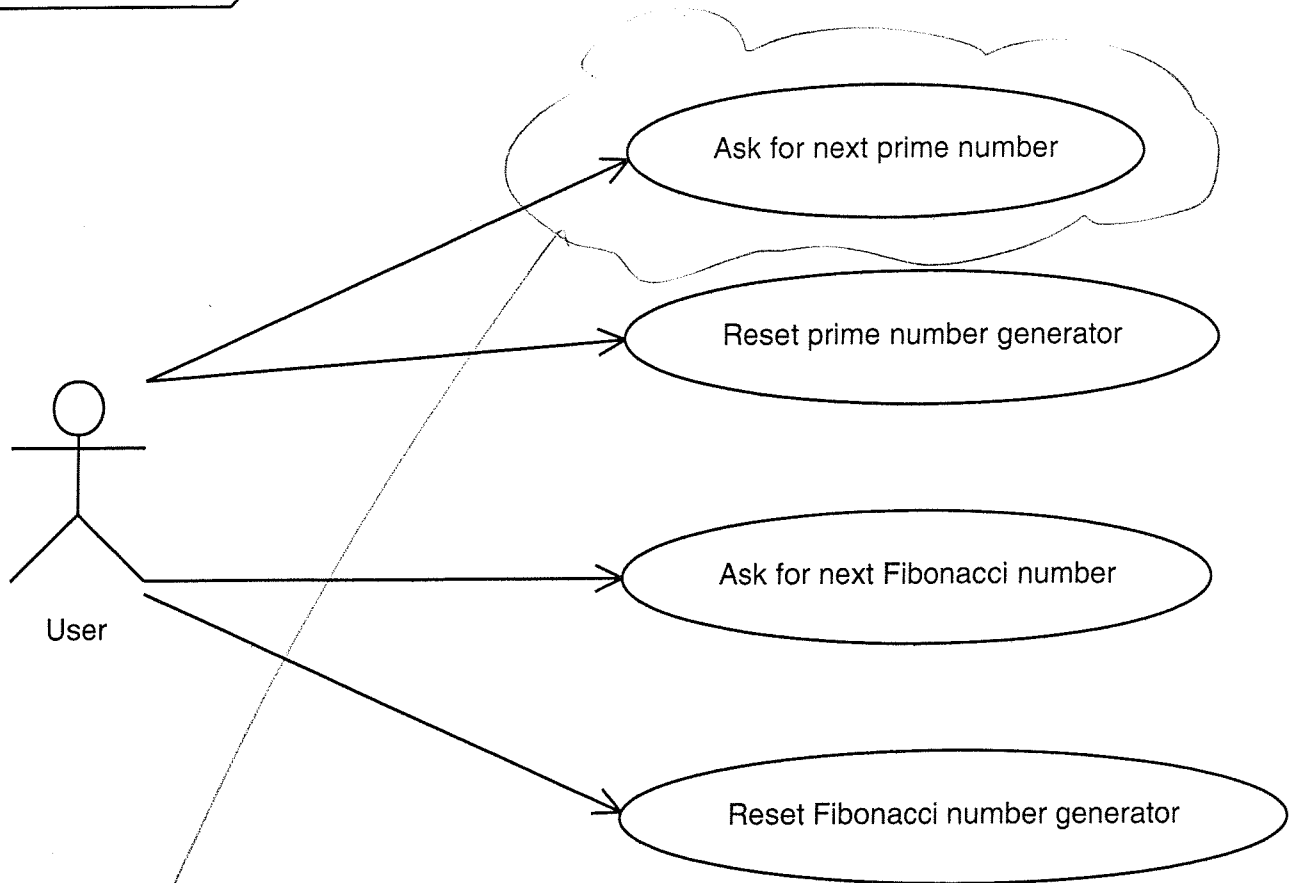
Use case: Ask for next Fibonacci number

1. **The use case starts here.**
2. The user pushes the button labelled "Next Fibonacci number" in the GUI.
3. The smallest Fibonacci number following the number that is displayed in the output field in the GUI is computed by the application.
4. The new number is displayed in the output field.
5. **The use case ends here.**

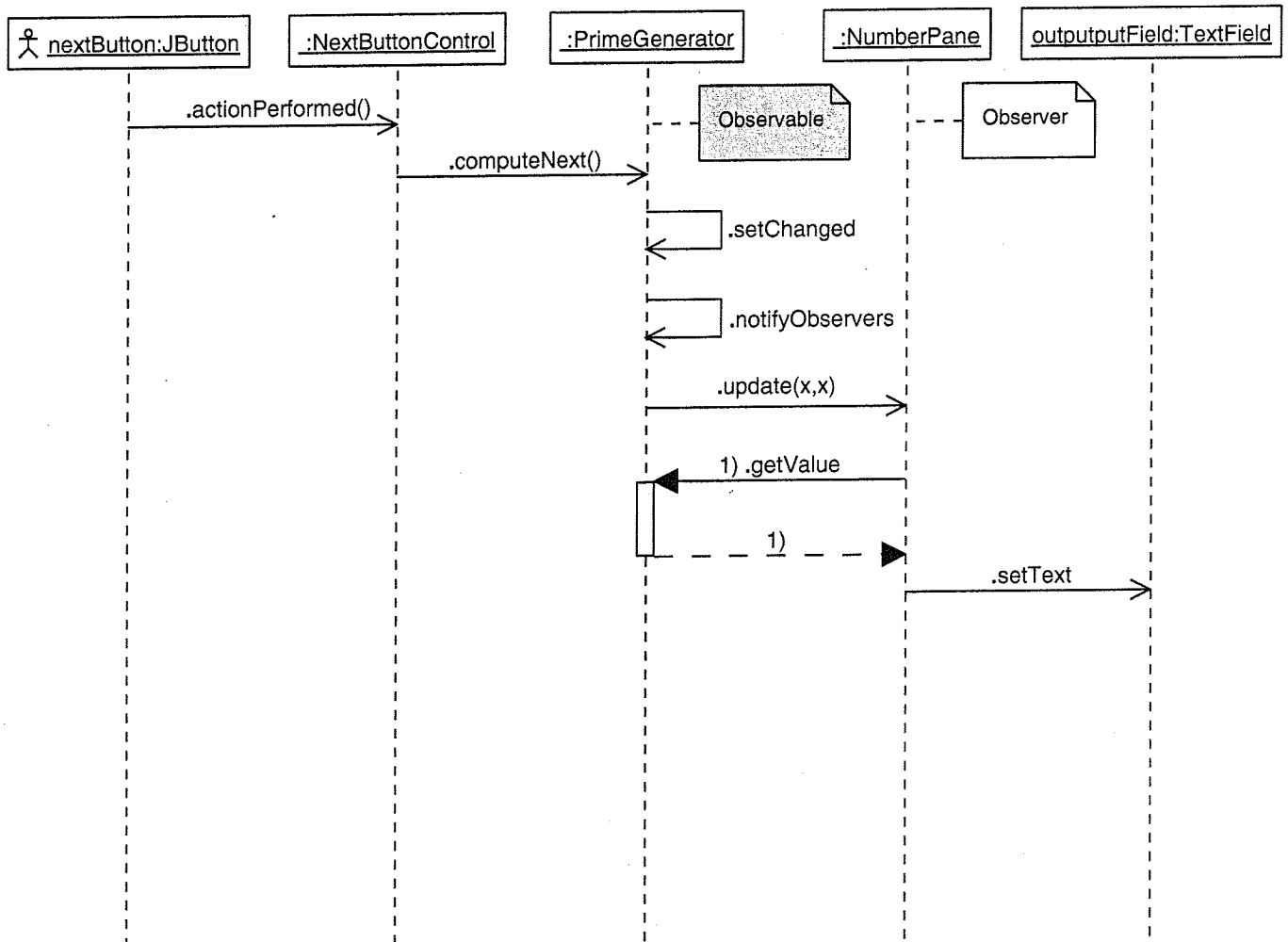
Use case: Reset Fibonacci number generator

1. **The use case starts here.**
2. The user pushes the Fibonacci number generator's "Reset" button in the GUI.
3. The smallest Fibonacci number is computed by the application.
4. The new number is showed in the output field.
5. **The use case ends here.**

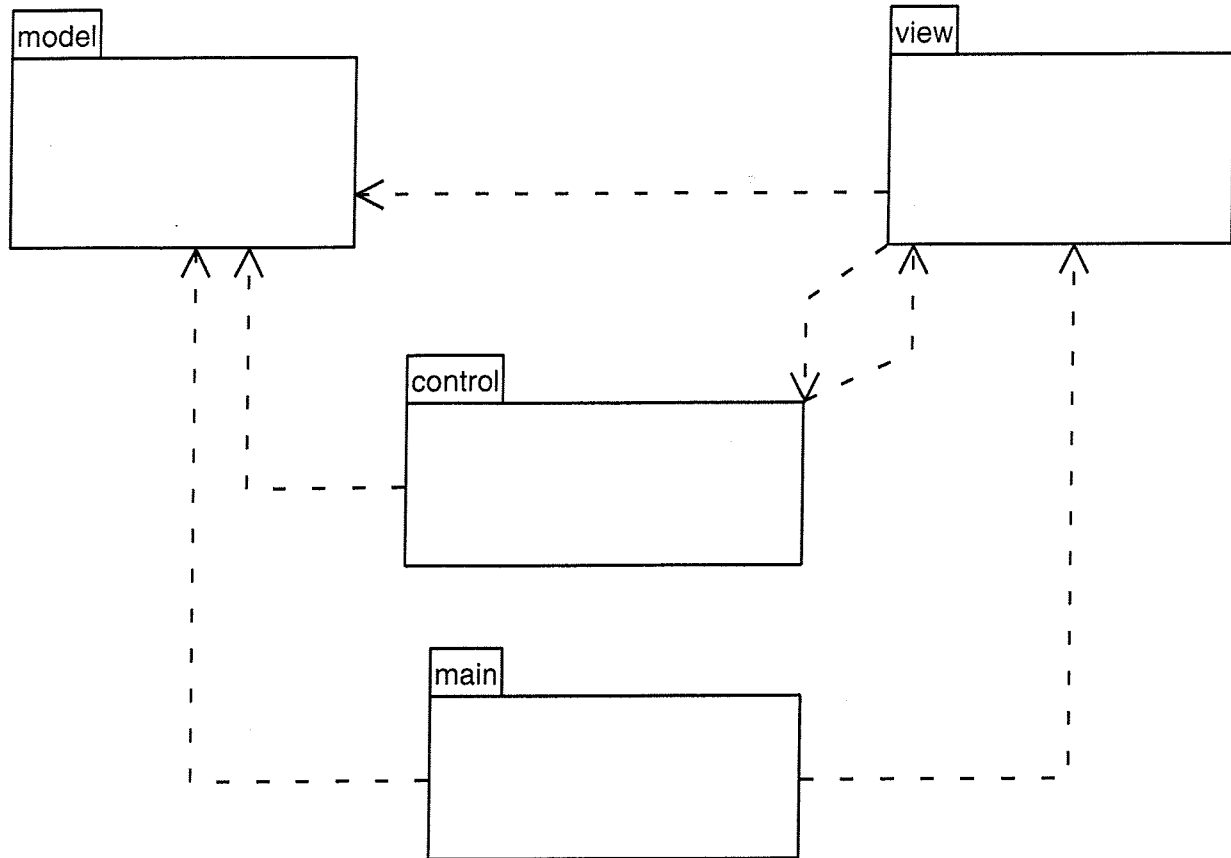
Use Case diagram_1



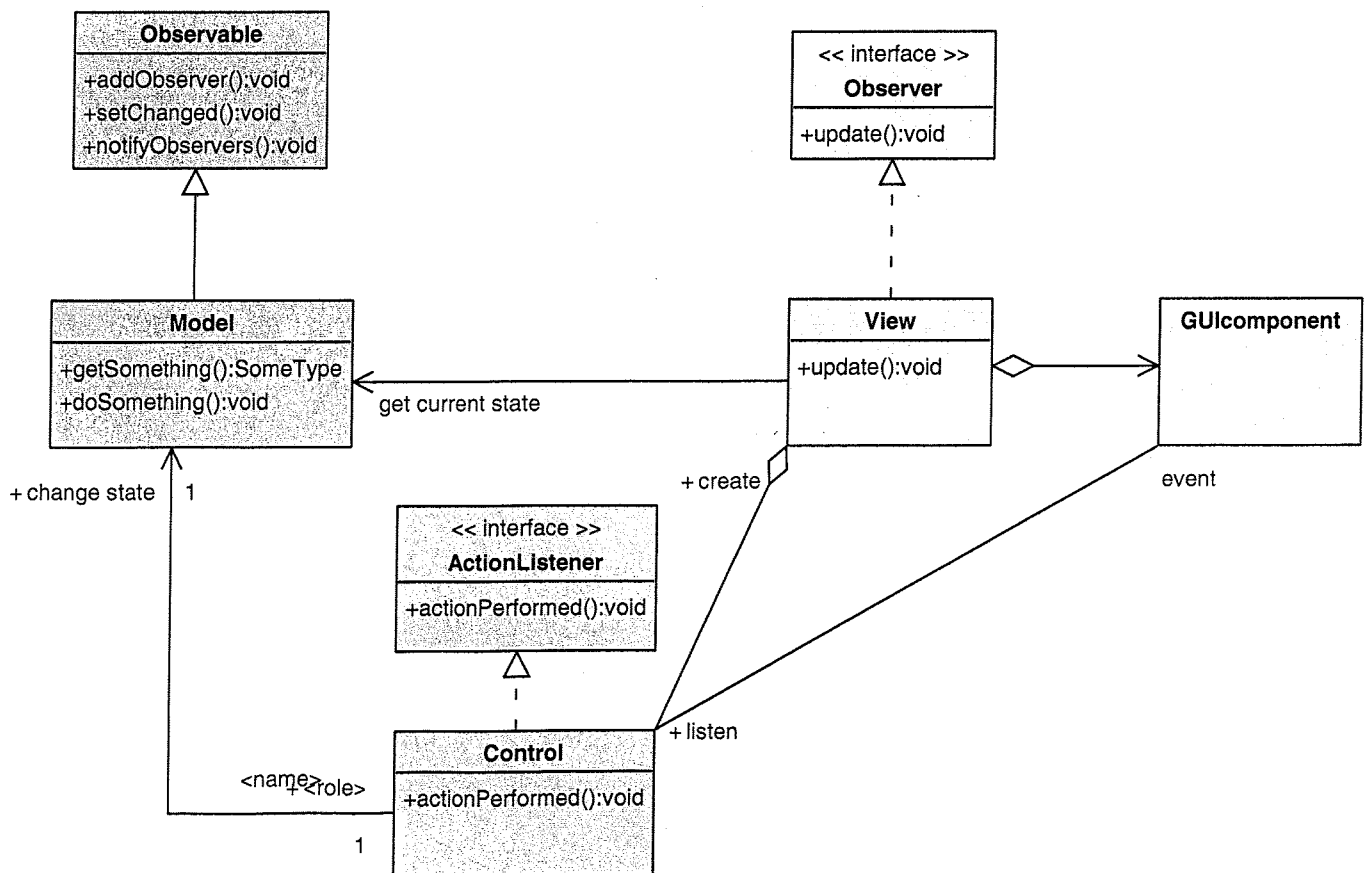
sd: Use case: Ask for next prime number



cd: Package Overview: MVC model

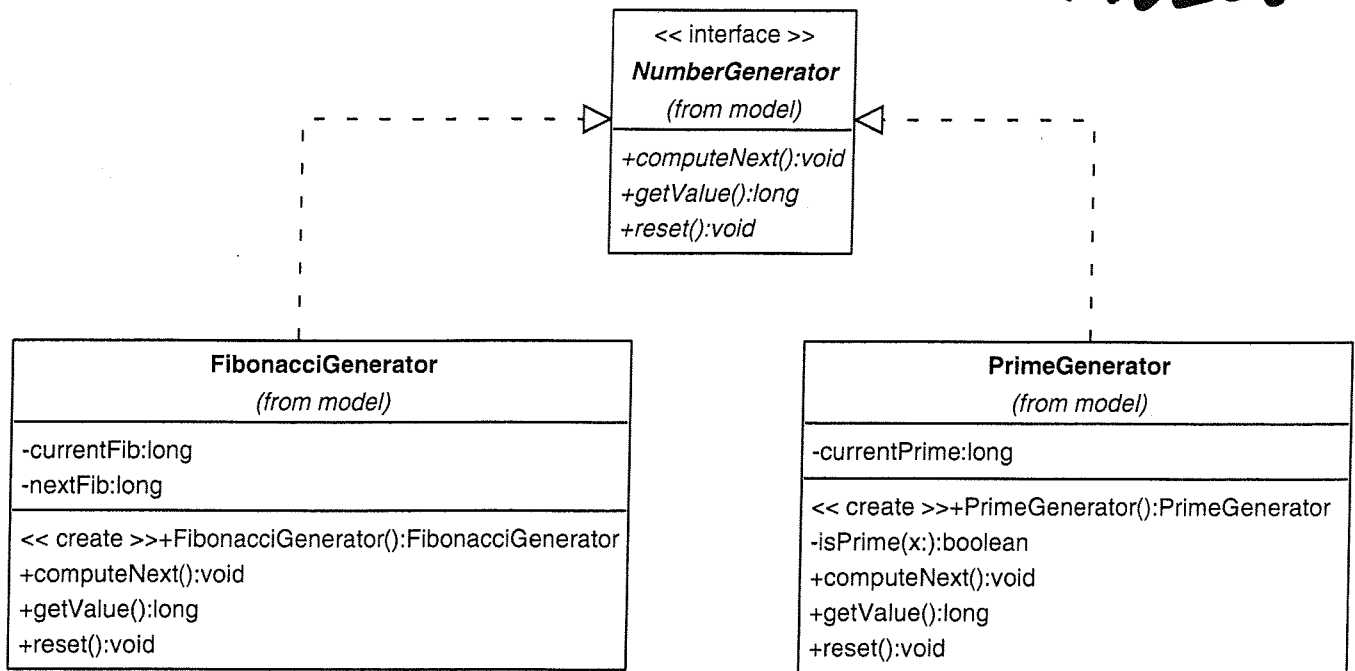


cd: MVC and the Observer model



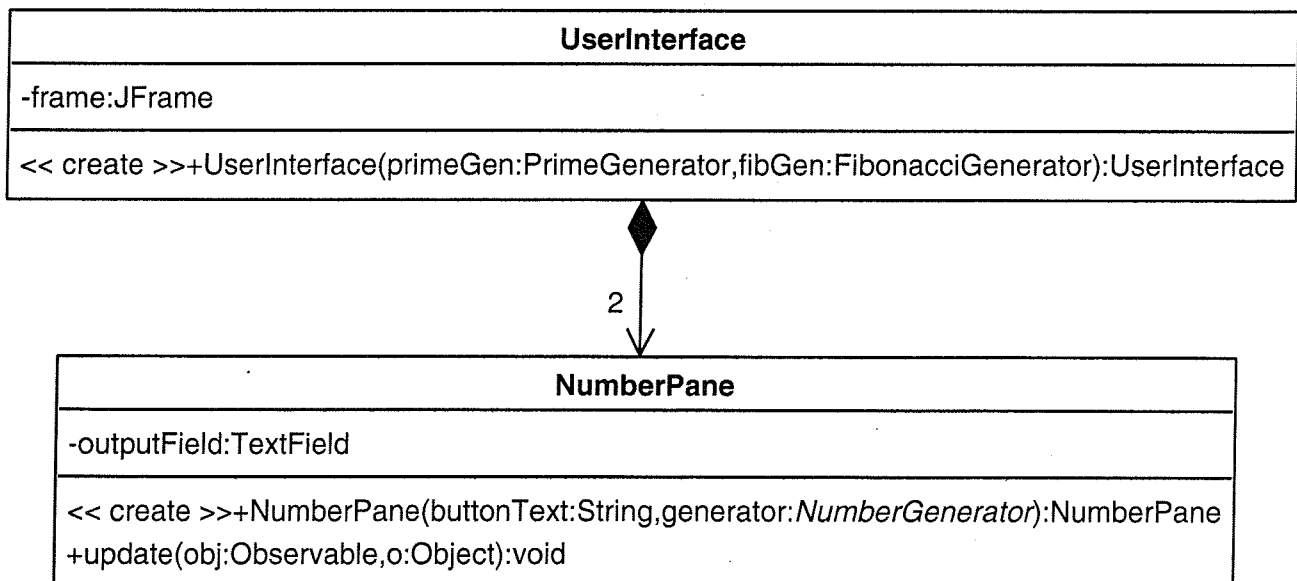
cd: Package Overview: model

Model



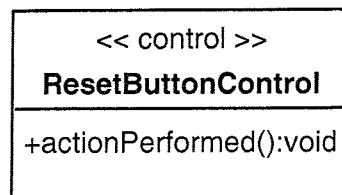
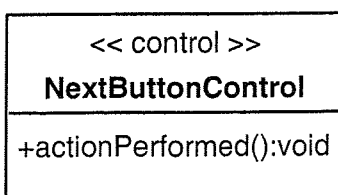
Package Overview: view

View

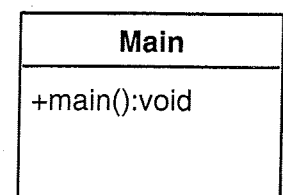


cd: Package overview: control

Control



cd: Package overview: main



Main