Introduction to object oriented programming in JAVA

Computer Applications in
Power Systems – Advance course

EH2750

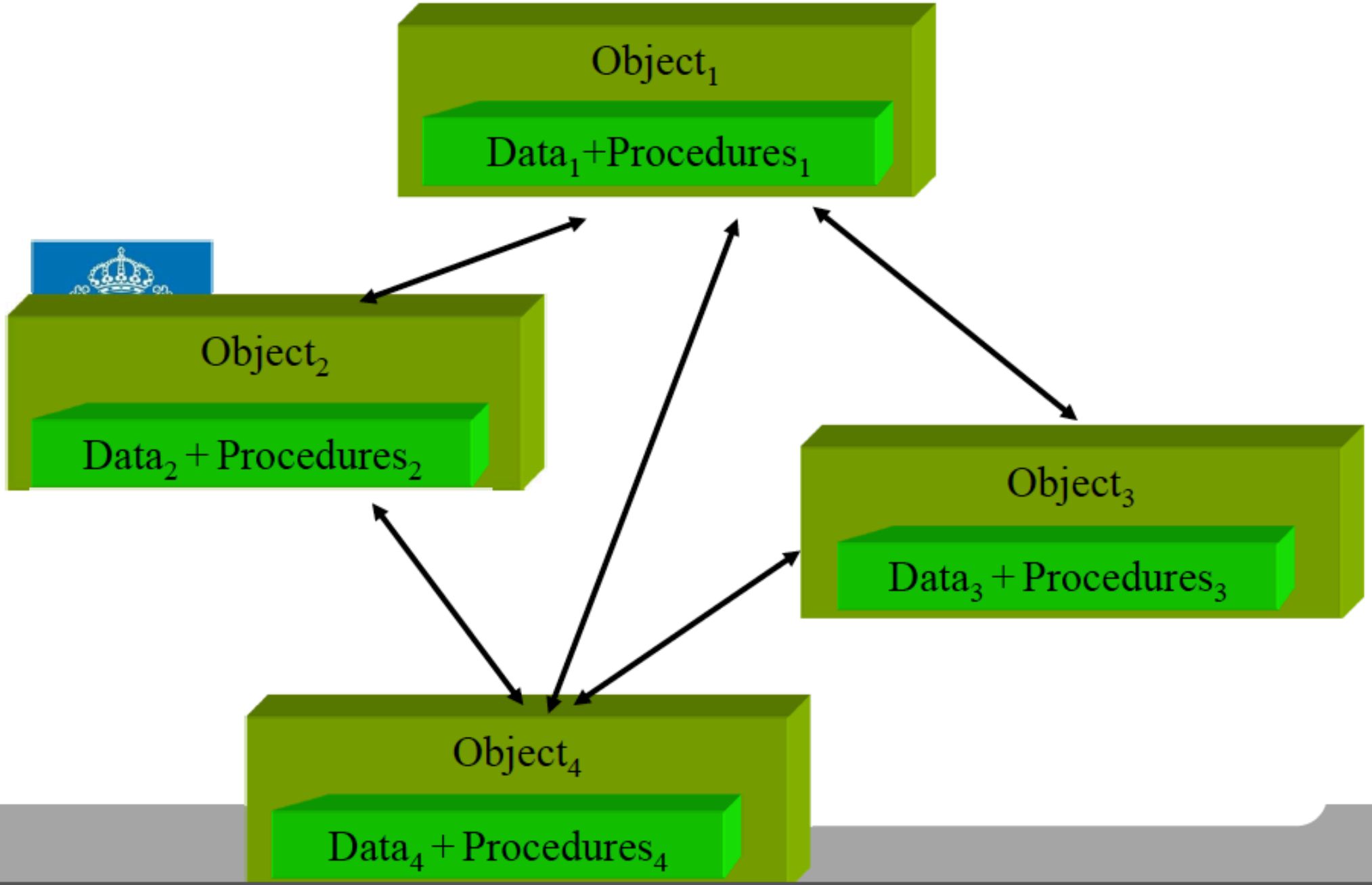
Dr. Arshad Saleem

Lecture material contribution from: John Lewis and William Loftus. Java Software Solutions, foundation of program design

contents

- Installation and setup
- OOP architecture revisit
- Programming in JAVA
- Working together
  - Hello World – the first program
  - Declaring variables and assignments
  - Implementing basic structures
    - Conditions and continuations
  - Implementing inheritance in JAVA
- Working individually/ in groups
  - Implementing a class diagram
  - Implementing class relationships
  - Implementing methods
  - Implementing use cases
  - HomeWork

OOP based system view



Java Program Structure

- A program is made up of one or more *classes*
- A class contains one or more *methods*
- A method contains program *statements*
- A Java application always executes the *main* method

HelloWorld.java

```
//HelloWorld, my first program in Java
class HelloWorld {

   public static void main(String[] args) {
      System.out.println("Hello World...")
      try{System.in.read();}
      catch(Exception e){}
   }// method main

}// class HelloWorld
```

The Java API

- The Java *Application Programmer Interface* (API) is a collection of classes that can be used as needed
- The `println` and `print` methods are part of the Java API; they are not part of the Java language itself
- Both methods print information to the screen; the difference is that `println` moves to the next line when done, but `print` does not

Importing Packages

- Using a class from the Java API can be accomplished by using its fully qualified name:

```
java.lang.System.out.println ();
```

- Or, the package can be imported using an *import statement*, which has two forms:

```
import java.applet.*;
import java.util.Random;
```

- The `java.lang` package is automatically imported into every Java program

Java Applets

- A *Java applet* is a Java program that is intended to be sent across a network and executed using a Web browser
- A *Java application* is a stand alone program
- Applications have a `main` method, but applets do not
- Applets are derived from the `java.applet.Applet` class
- See `Confucius.java` and `No_Parking.java`
- Links to applets can be embedded in HTML documents

Variables

- A *variable* is an identifier that represents a location in memory that holds a particular type of data
- Variables must be declared before they can be used
- The syntax of a variable declaration is:

```
data-type variable-name;
```

- For example:

```
int total;
```

Variables

- Multiple variables can be declared on the same line:

```
int total, count, sum;
```

- Variables can be *initialized* (given an initial value) in the declaration:

```
int total = 0, count = 20;
float unit_price = 57.25;
```

- See `Piano_Keys.java`

Assignment Statements

- An assignment statement takes the following form:

```
variable-name = expression;
```

- The expression is evaluated and the result is stored in the variable, overwriting the value currently stored in the variable
- See `United_States.java`
- The expression can be a single value or a more complicated calculation

Constants

- A constant is similar to a variable except that they keep the same value throughout their existence
- They are specified using the reserved word `final` in the declaration
- For example:

```
final double PI = 3.14159;
final int STUDENTS = 25;
```

Constants

- When appropriate, constants are better than variables because:
  - they prevent inadvertent errors because their value cannot change
- They are better than literal values because:
  - they make code more readable by giving meaning to a value
  - they facilitate change because the value is only specified in one place

BasicMath.java


```
class BasicMath {
public static void main (String[] args) {
final double PI = 3.14159;
int radius = 6;
double area, circum;
circum = PI * radius * 2;
area = PI * radius * radius;
System.out.println("Circumference is: " + circum);
System.out.println("Area is: " + area);
try{System.in.read();}
catch(Exception e){}
}//method main
}//class BasicMath
```

```
Circumference is: 37.699079999999995
Area is: 113.09723999999999
```

Input and Output

- Java I/O is based on *input streams* and *output streams*
- There are three predefined standard streams
- The `print` and `println` methods write to standard output

Stream	Purpose	Default Device
`System.in`	reading input	keyboard
`System.out`	writing output	monitor
`System.err`	writing errors	monitor

Input and Output

- The Java API allows you to create many kinds of streams to perform various kinds of I/O
- To read character strings, we will convert the `System.in` stream to another kind of stream using:

```
   BufferedReader stdin = new
BufferedReader

      (new InputStreamReader (System.in),
1);
```

- This declaration creates a new stream called `stdin`

Echo.java

```
import java.io.*;
class Echo {
public static void main (String[] args) throws IOException {
    BufferedReader stdin = new BufferedReader
        (new InputStreamReader(System.in),1);
    String message;

    System.out.println ("Enter a line of text: ");
    message=stdin.readLine();
System.out.println ("You entered: \"" + message + "\"");
    }
}
```

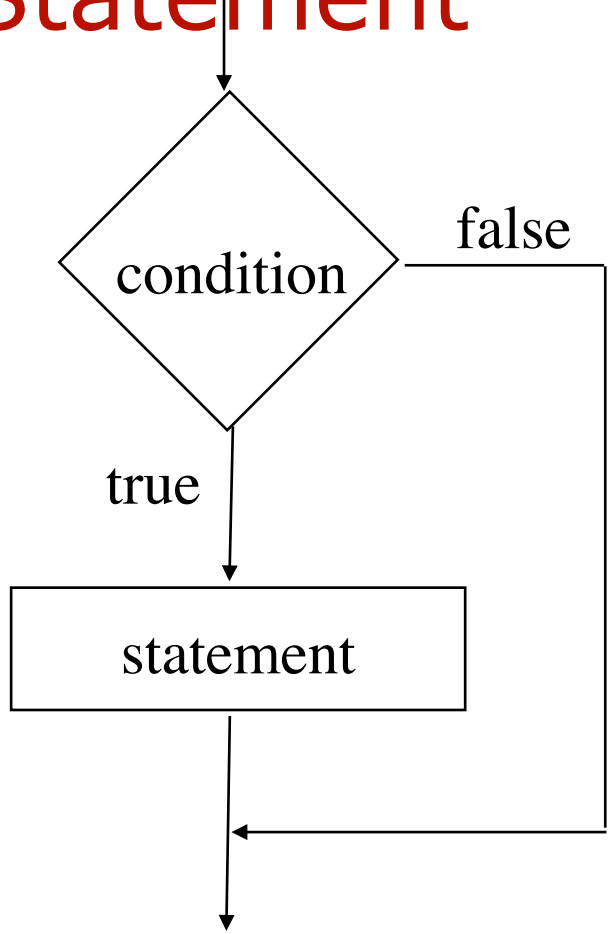
The if Statement

- The Java *if statement* has the following syntax:

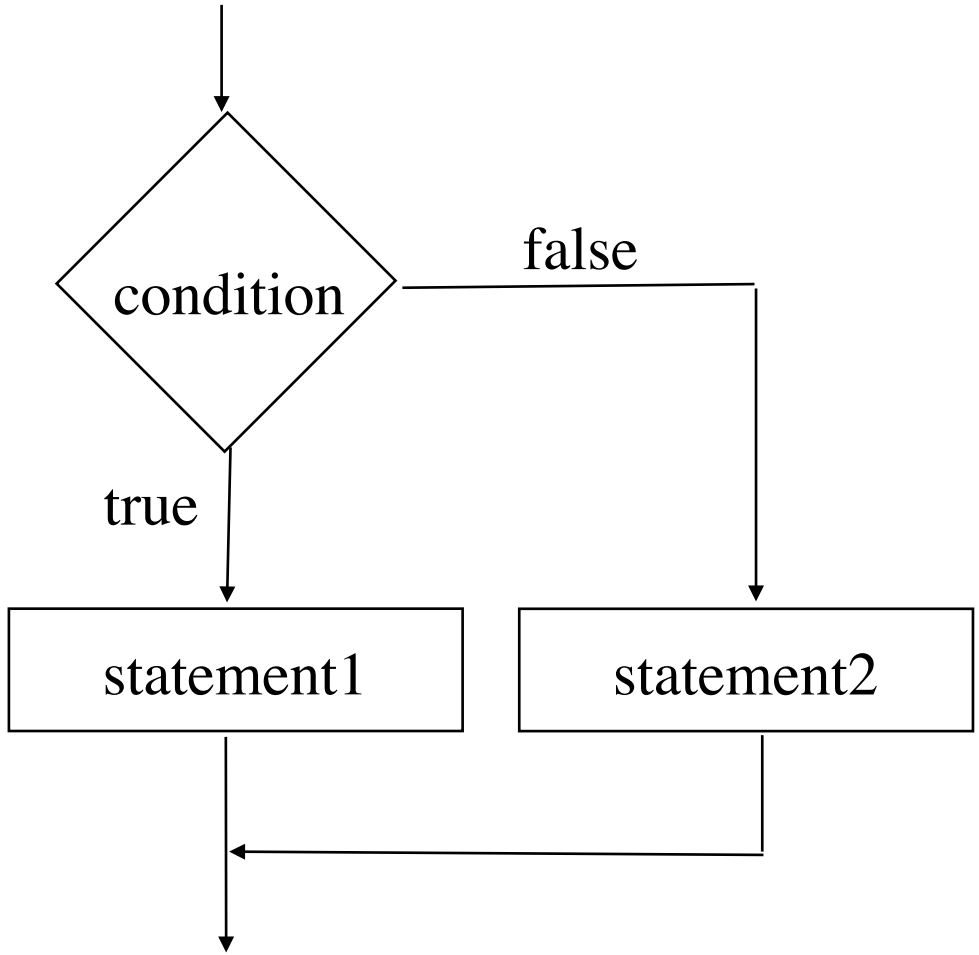
```
if (condition)
    statement;
```

- If the boolean condition is true, the statement is executed; if it is false, the statement is skipped
- This provides basic decision making capabilities

The if Statement



The if-else Statement



BankBalance.java

```
if (withdrawal > balance)
System.out.println("Don't got that much cash!");
else
if ((balance - withdrawal) < 150)
System.out.println("New balance " +
(balance - withdrawal) + " is below $150");
else
System.out.println("New Balance is: " +
(balance - withdrawal));
}// method main
}// class BankBalance
```

Repetition Statements

- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*
- The programmer should choose the right kind of loop for the situation

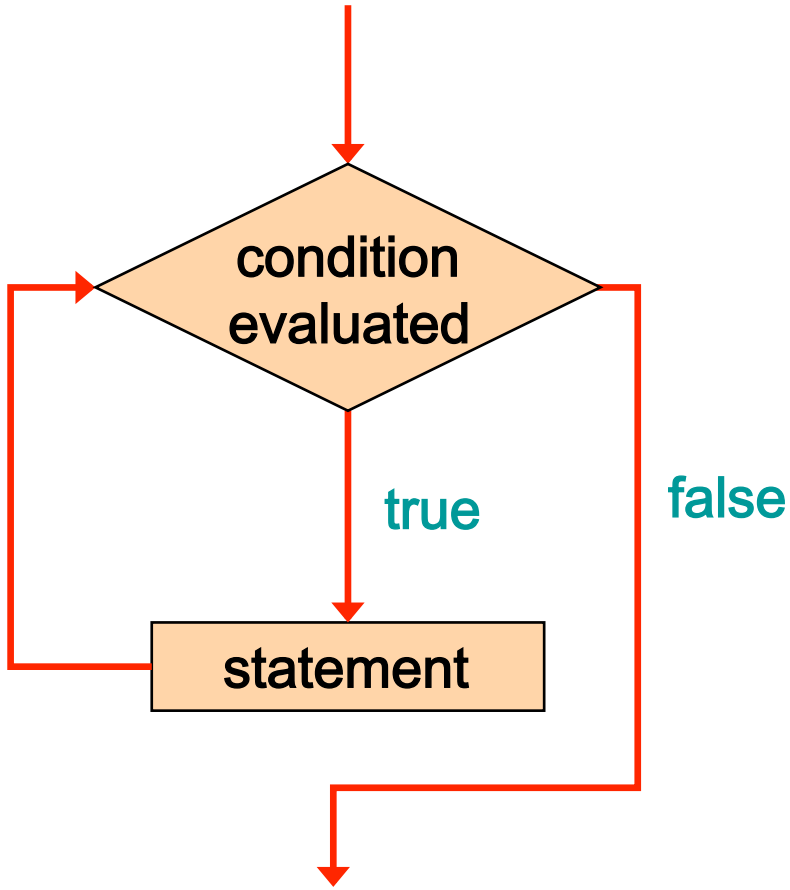
The while Statement

- A *while* *statement* has the following syntax:

```
while ( condition )
      statement;
```

- **If the `condition` is true, the `statement` is executed**
- **Then the condition is evaluated again, and if it is still true, the statement is executed again**
- **The statement is executed repeatedly until the condition becomes false**

Logic of a while Loop



The while Statement

- An example of a while statement:

```
int count = 1;
while (count <= 5)
        {
System.out.println (count);
        count++;
        }
```

- **If the condition of a `while` loop is false initially, the statement is never executed**
- **Therefore, the body of a `while` loop will execute zero or more times**
- ***** how to make an infinite loop?**

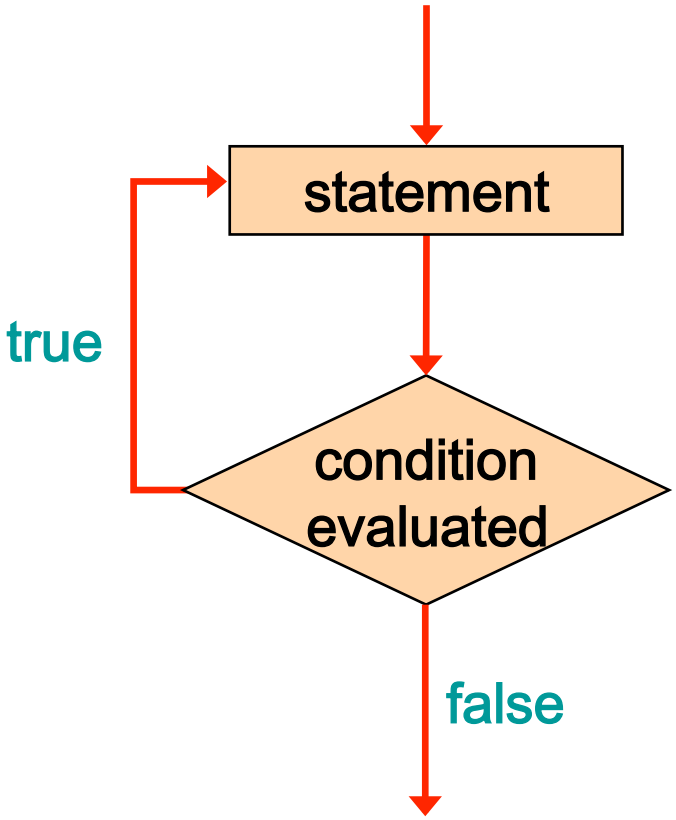
The do Statement

- A *do statement* has the following syntax:

```
do
{
statement;
}
while ( condition );
```

- **The `statement` is executed once initially, and then the `condition` is evaluated**
- **The statement is executed repeatedly until the condition becomes false**

Logic of a do Loop



The do Statement

- An example of a `do` loop:

```
int count = 0;
do
{
count++;
System.out.println (count);
} while (count < 5);
```

- **The body of a `do` loop executes at least once**
- **See ReverseNumber.java (page 244) Listing 5.12**

The for Statement

- A *for statement* has the following syntax:

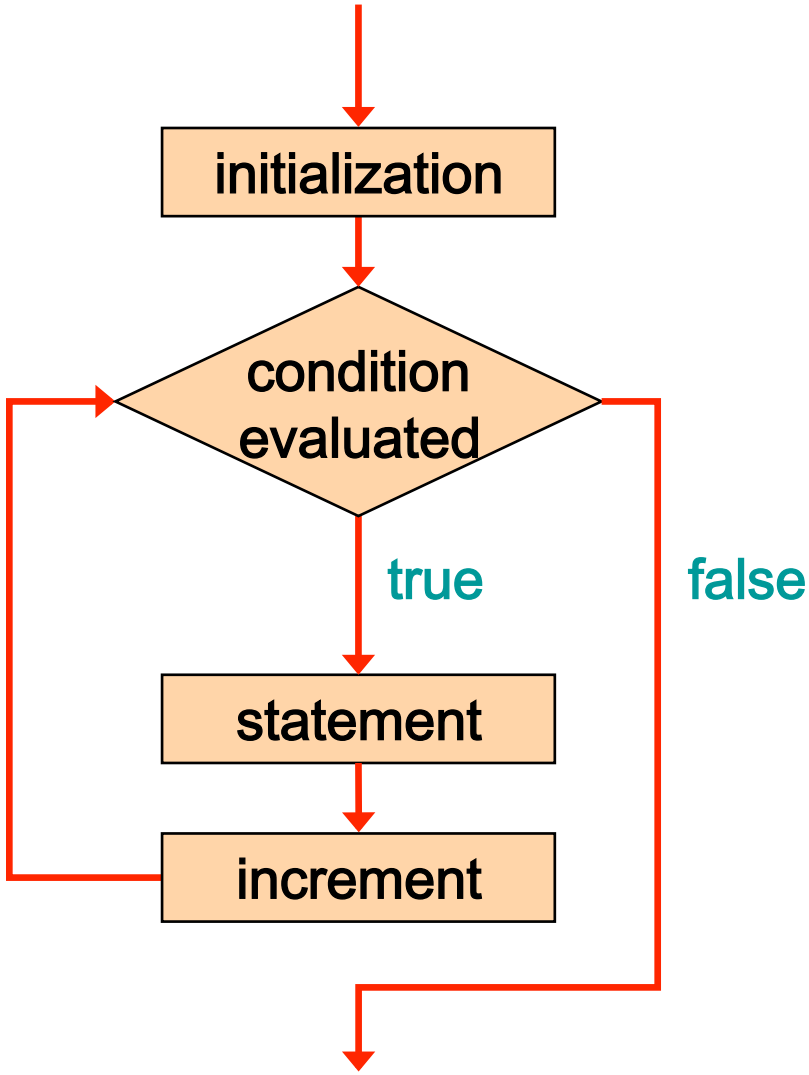
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
    statement;
```

The *increment* portion is executed at the end of each iteration

Logic of a for loop



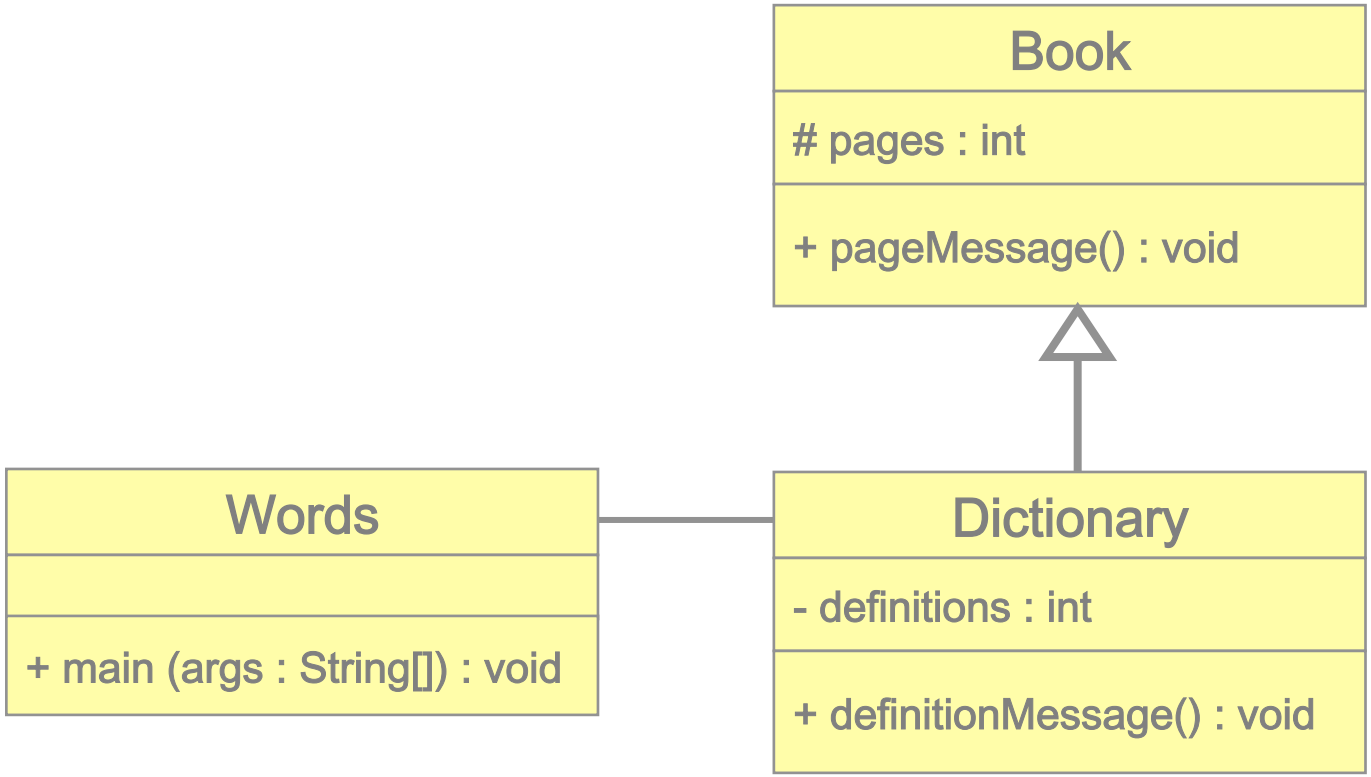
The for Statement

- An example of a `for` loop:

```
for (int count=1; count <= 5; count++)
    System.out.println (count);
```

- **The initialization section can be used to declare a variable**
- **Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body**
- **Therefore, the body of a `for` loop will execute zero or more times**

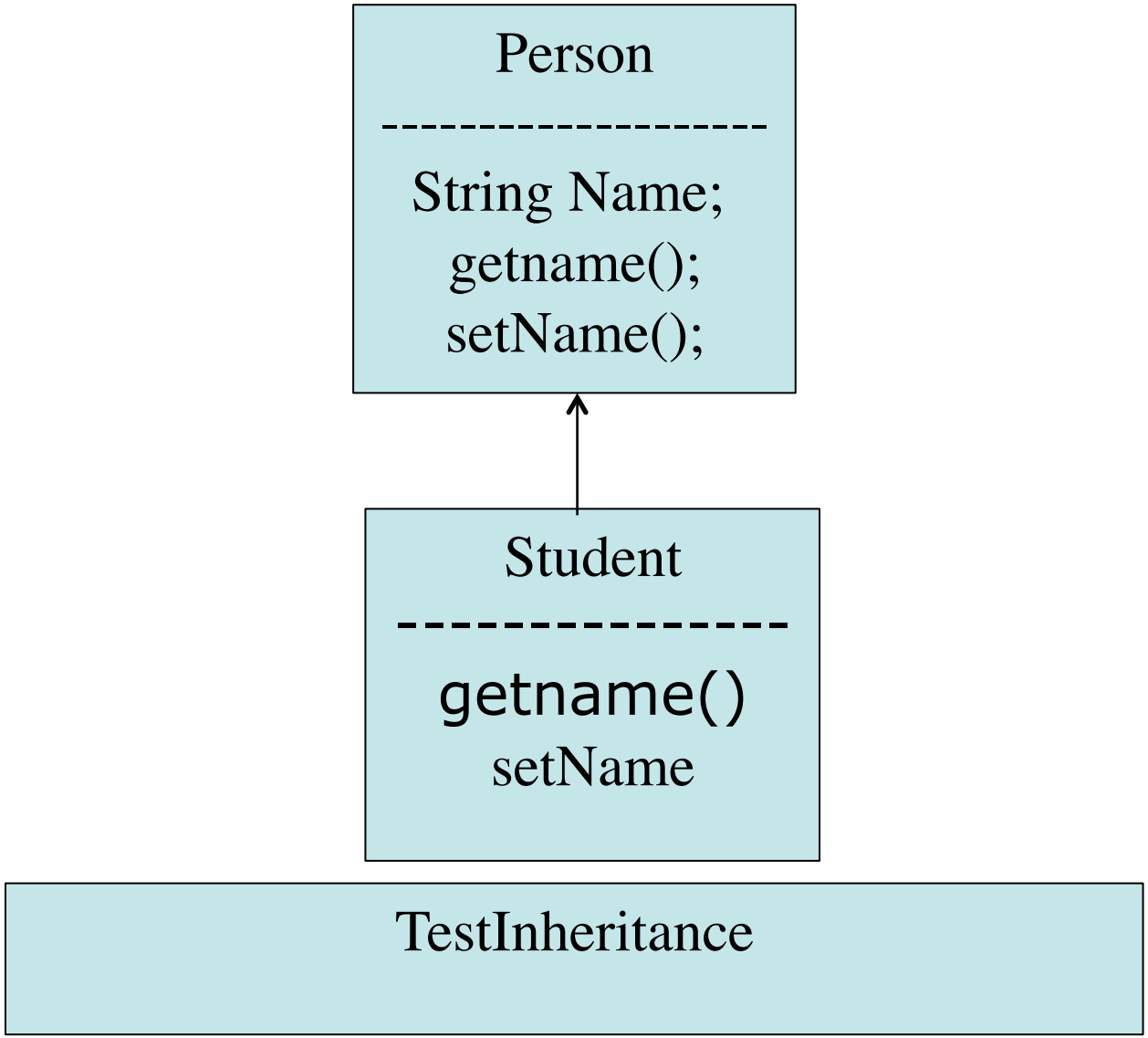
Inheritance example



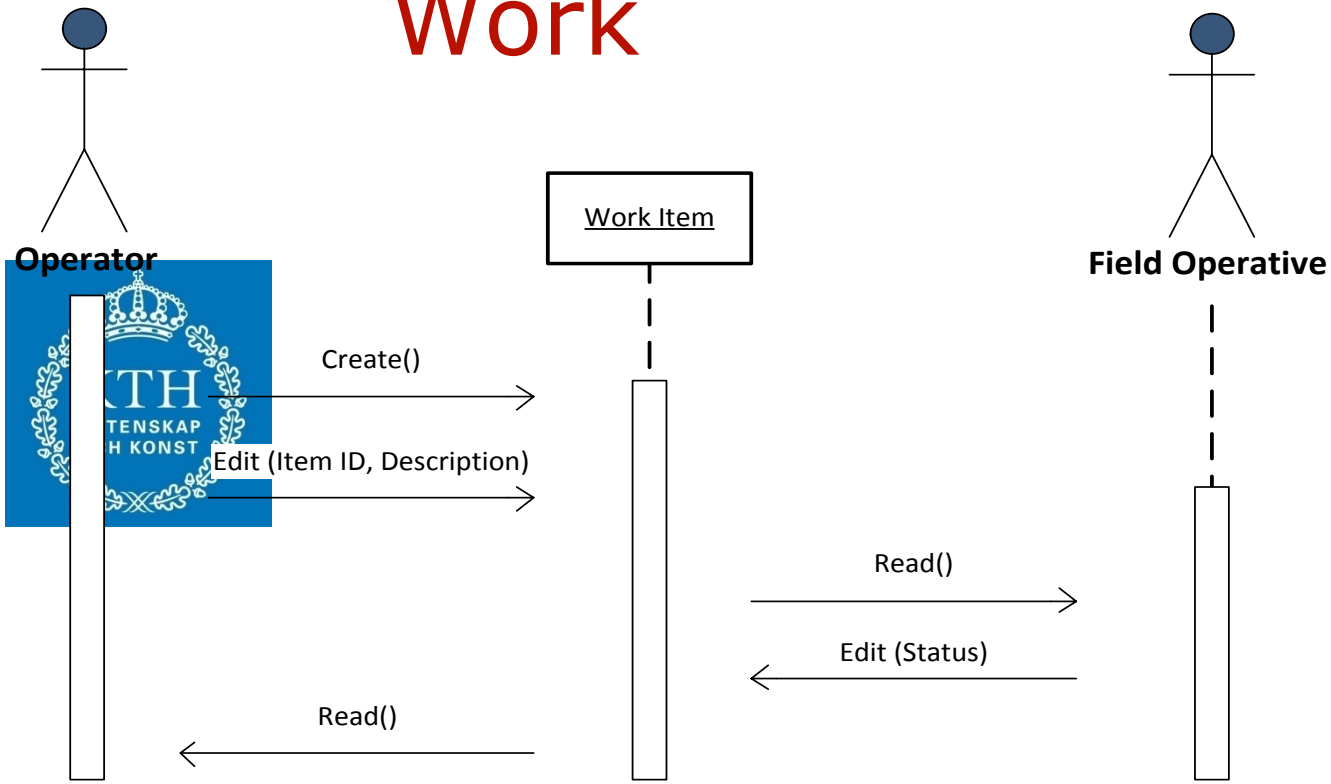
Inheritance example

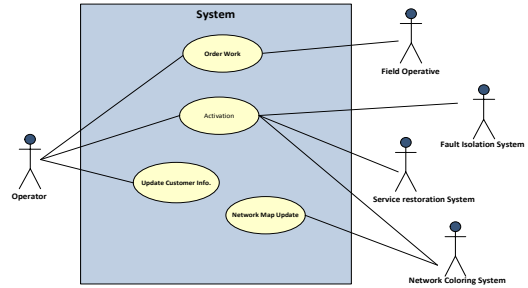

```
package MyPackage;

class Base {
     private int x;
     public int f() { ... }
     protected int g() { ... }
}

class Derived extends Base {
     private int y;
     public int f() { /* new implementation for Base.f() */ }
     public void h() { y = g(); ... }
}
```

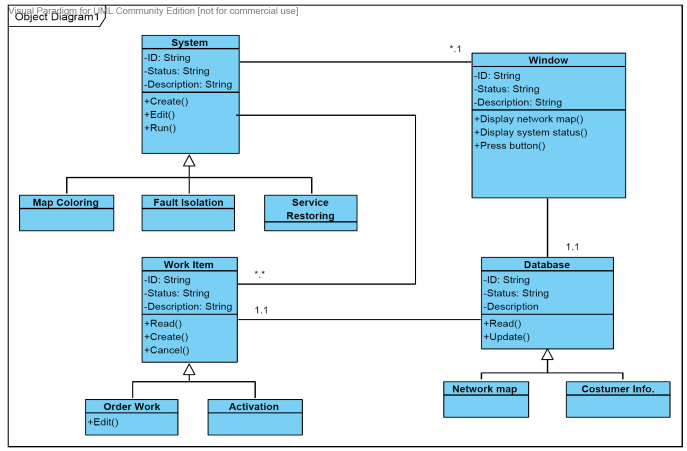
Implementing classes and inheritance in JAVA



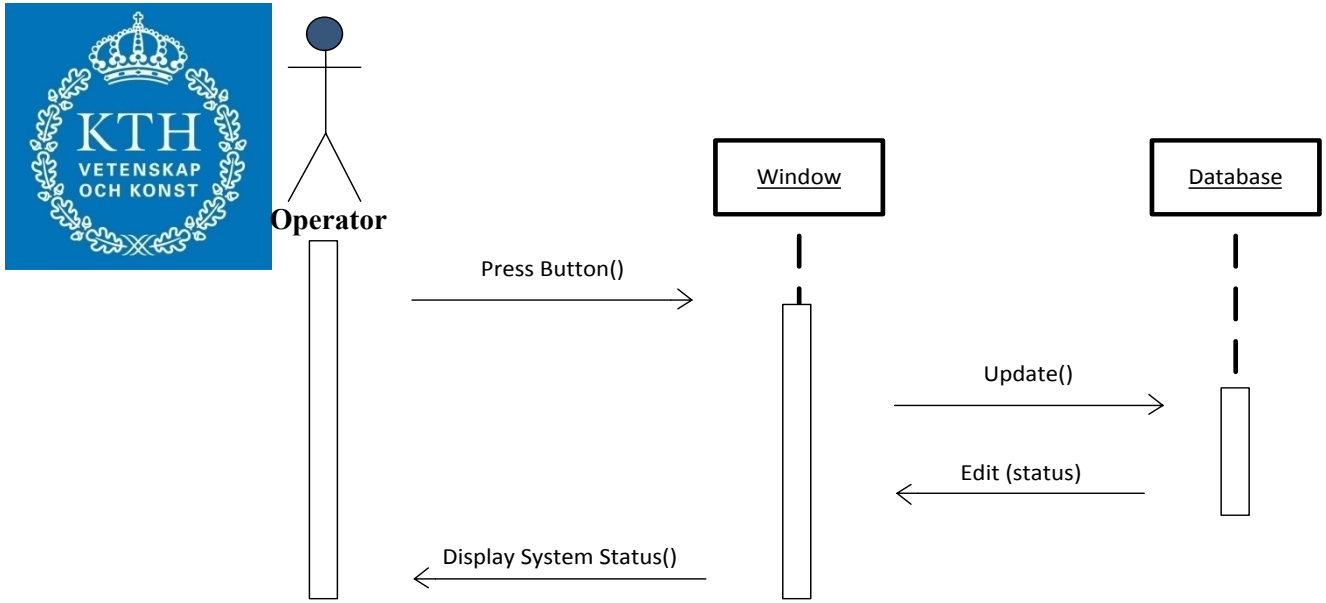
Sequence Diagram - Order Work

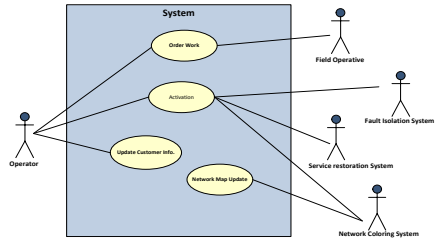


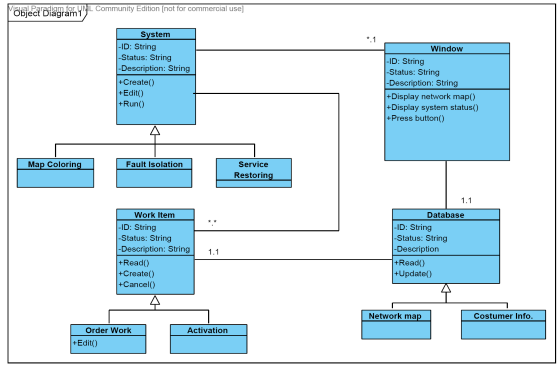




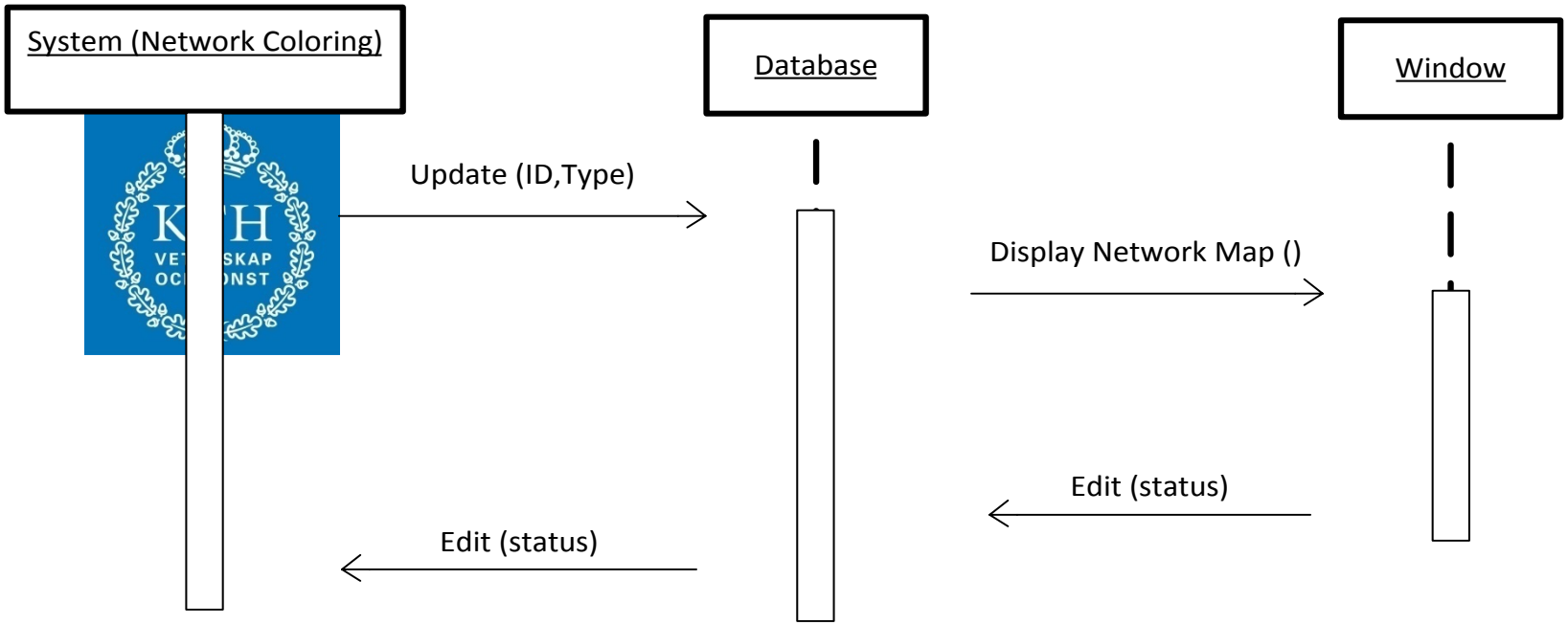
Sequence Diagram - Update Costumer Info

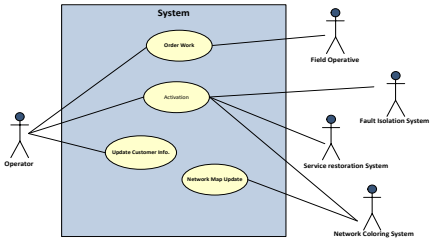


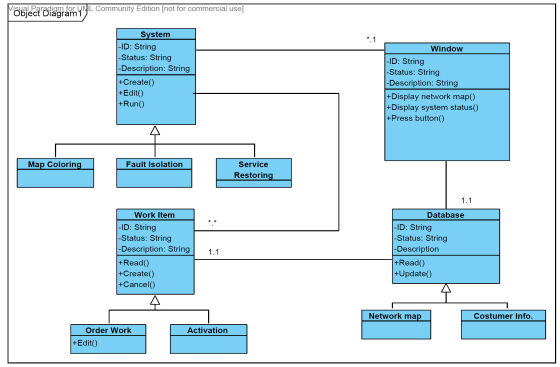




Sequence Diagram - Network Map Update







Home Work

- Implement in JAVA your usecase from 1st assignment