Programming I

Events

12th lecture

Overview

- AHA!

Event

- What are events?
  - open doors
  - light comes on
  - start of a new minute
  - mouse moves
  - boat cames to the other bank
  - an error has occurred
- What can we say about these events?
  - When they may occur in the course of the program? ever?
  - What is the last event?

Interrupts

- Interrupts (prekinitve) are special types of events in computing.
- Triggered by unexpected (events) when something happened:
  - the computer - ran out of power,
  - operating system - has ended a task or in the environment of the program - someone moved the mouse

Events in CS

- We distinguish between interrupts caused by hardware and interrupts caused by software (hardware and software interrupts).
- Hardware interrupt: moving your mouse, keystroke, a new millisecond has passed, etc..

Events in CS

- Software interrupts: division by zero, out of memory, someone is trying to access a field that does not exist, etc..
- A special type of program termination messages are on special (exceptional) situations – Exceptions (at the end of the lecture).

What to do at an event

- When an event occurs, it is necessary to react.
- We say that it is necessary to process an event or handle it.
- Therefore, we call the functions, objects, etc.., which process events: **an event handler**.

Events handling

- Any event can be handled by one or more handlers.
- Handler is connected to an object where events can happen.
- Handler can also disconnected (removed - uninstalled).

Event handler

- Event handler is an object (with special properties - methods)
- There must exist an understanding thar a special method exists – DO_THE_WORK
- The object event handler is introduced to the event, the special method DO_THE_WORK starts when the event happens

Event handler

```
public interface Handler {
  public void do();
}
                                    ...
public class EventHandler implements Handler{
  public void do(void) {
    System.out.println(“An event happened“);
  }
}
EventHandler h = new EventHandler();
someObject.install(h);
```

Example – input stream

- Consider a class Reader who will read the letters from the input stream
- Let the letters be from some set A called the alphabet
- The events that the reader class distinguishes are the individual read letters:
  - let A = {0, 1, 2}
  - then the events are: read 0, read 1, ...
- A handler can be installed for each of the events

Razred *bralec* in dogodki

- To simplify the installation of handlers at different events, we have only one installation method:
  - `void install` (deadline handler, char letter);
- which installs a handler on the event that the character »letter« was read,
- the class also has a read method that triggers the reading of the input data stream,
- The input is read from standard input stream.

Interface Reader

```
public interface Reader {

  public void install(Handler h, char character);
  public void read();

};
```

Preštejmo *a*-je

- Using the reader class, we want to count the number of letters ’a’ in the input data stream.
- We define a class stejA which:
- is the implementation of the handler interface
- the handling method increases the counter by 1 for each call
- the method will be called when the reader reads the letter a
- has an additional number method that returns the current value of the counter

Class Count*A*

```
public class CountA implements Handler {

  private int count = 0;

  public      CountA()   { count = 0;     }
  public void do() { count++;      };
  public int  number() { return count; };
}
```

Usage

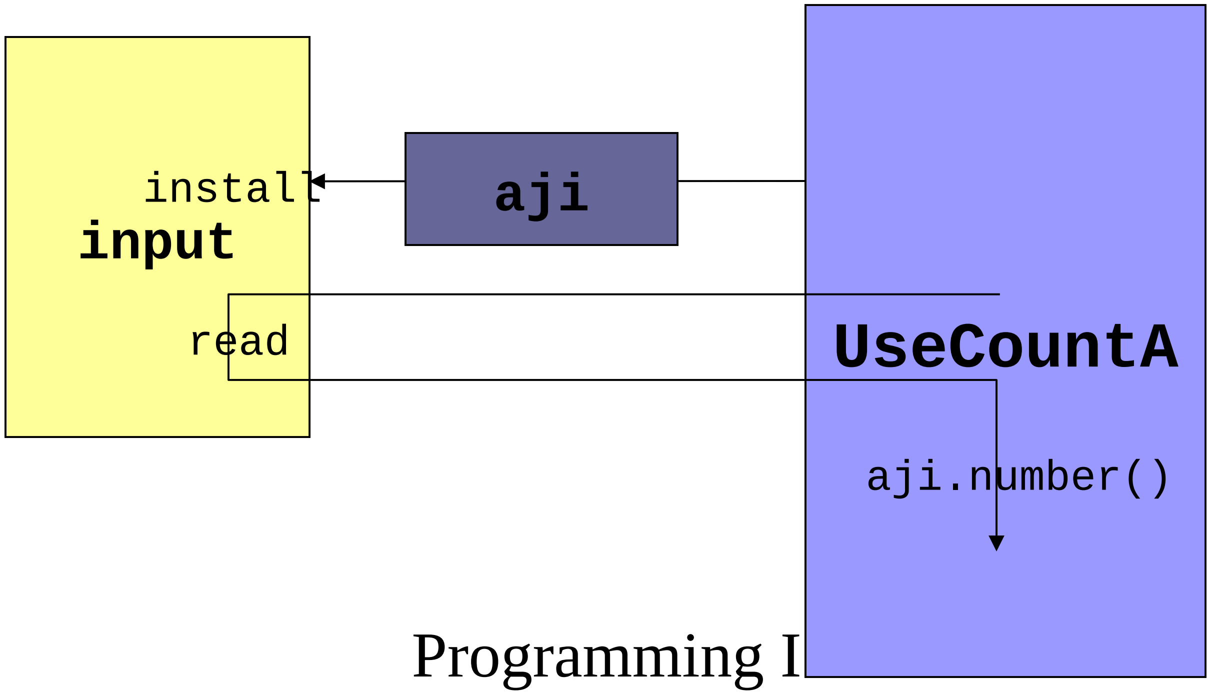
```
public class UseCountA {

  public static void main (...) {
    CountA  aji =  new CountA();
    Reader input = new Reader();

    input.install(aji, 'a');
    input.read();
    System.out.print("Number of a: ");
    System.out.print( aji.count() );
    System.out.println();
  } // main

} // UporabastejA
```

Arhitektura sistema



Event handler for special situations

```
try {
   ...;
} catch (Exception message) {
   System.out.println("something happened " +
                                      message);
};
```

Exceptions

- The term exception is shorthand for the phrase "exceptional event".
- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

The Try/Catch statement

- Code that might throw certain exceptions must be enclosed by:
  - A try statement that catches the exception.
  - A method that specifies that it can throw the exception.
- Code that fails to honor the Try/Catch or throw Requirement will not compile.

How to Throw Exceptions

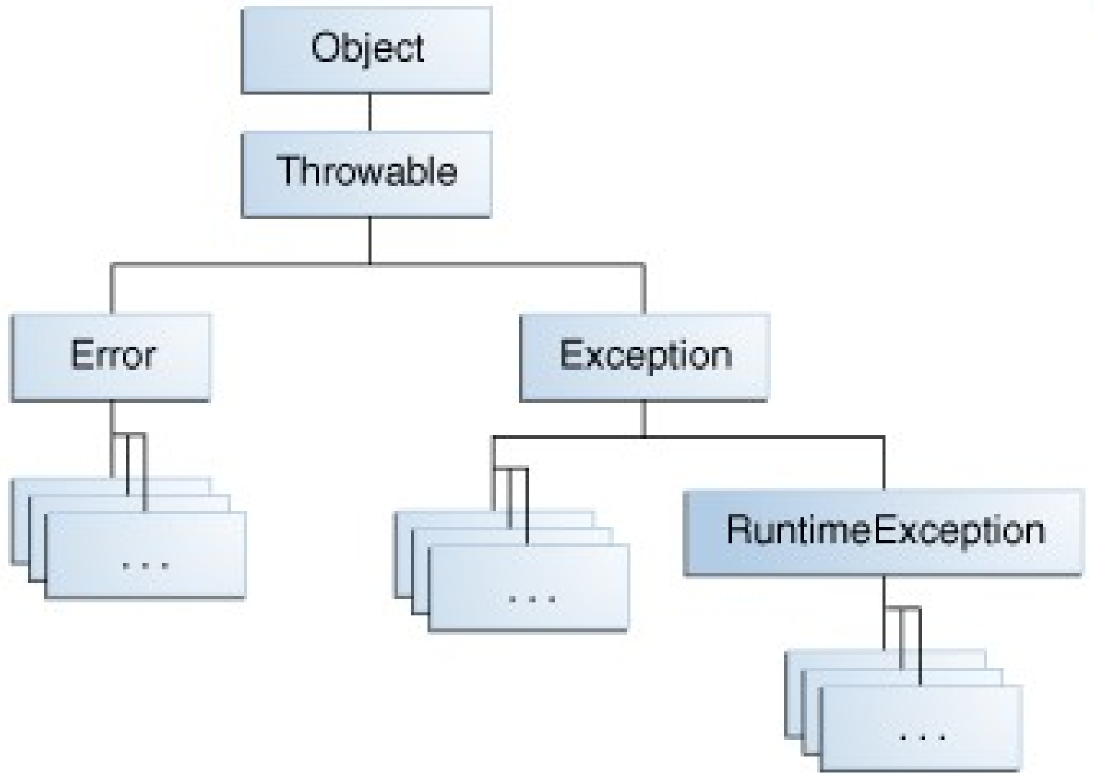
- Before you can catch an exception, some code somewhere must throw one.
  - Throw statement.

```
throw someThrowableObject;
```

How to Throw Exceptions

```
public Object pop() {
  Object obj;
  if (size == 0) {
   throw new EmptyStackException();
  }
  obj = objectAt(size - 1); setObjectAt(size -
  1, null);
  size--;
   return obj;
}
```

Throwable Class and Its Subclasses



Catching and Handling Exceptions

- You associate exception handlers with a try block by providing one or more catch blocks:

```
try {
  …
} catch (ExceptionType name) {
  …
} catch (ExceptionType name) {
  …
}
```

At the end

- The newly presented technique is called: **event driven programming**.
- This is the basis for real-time systems programming:
  - We must respond (as soon as) something (an event) happens.
  - An event triggers a reaction (action)
  - The reaction can also be a new event that triggers a new reaction …
- **Challenge:** how to make two event handlers cooperate?