

Real-Time Systems Programming



Summer-Semester 2002

Lecture 16

7 June 2002



Exceptions contd.



- 1) Exception handling in Ada, Java and C
- 2) Recovery blocks and exceptions
- 3) Exceptions and timing faults

Exception Handling in Ada

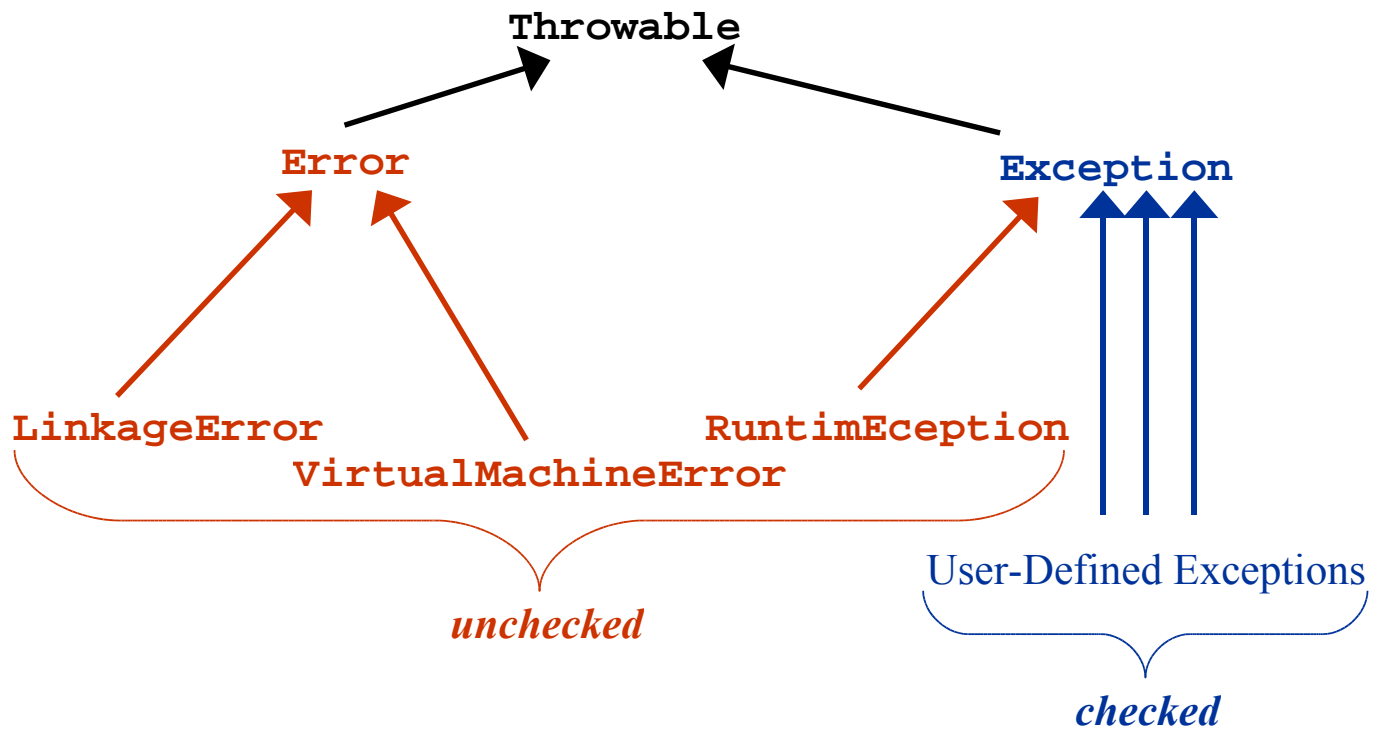


- Ada supports:
 - Explicit exception declaration
 - The termination model
 - Propagation of unhandled exceptions
 - A limited form of exception parameters
- *For further information:*
 - See Chapters 6 and 12 of [Burns and Wellings 2001]



- ... also use termination model
- ... however, are integrated into OO model
- ... are subclasses of the predefined class **java.lang.Throwable**
- The language also defines other classes, for example: **Error**, **Exception**, and **RuntimeException**
 - RuntimeException includes **ClassCastException**, **IndexOutOfBoundsException**, **NullPointerException**, etc.
- **Checked** exceptions have to be identified in in the **throws** clause of a method's declaration

The Throwable Class Hierarchy



Example: Temperature Controller



```
public class IntegerConstraintError extends Exception
{
    private int lowerRange, upperRange, value;

    public IntegerConstraintError(int L, int U, int V)
    {
        super();          // call constructor on parent class
        lowerRange = L;
        upperRange = U;
        value = V;
    }

    public String getMessage()
    {
        return ("Integer Constraint Error: Lower Range " +
            java.lang.Integer.toString(lowerRange) + " Upper Range "
            + java.lang.Integer.toString(upperRange) + " Found " +
            java.lang.Integer.toString(value));
    }
}
```

Temperature Controller cont.



```
import exceptionLibrary.IntegerConstraintError;

public class Temperature
{
    private int T;

    public Temperature(int initial) throws IntegerConstraintError
        // constructor
    { ... }

    public void setValue(int V) throws IntegerConstraintError
    { ... }

    public int readValue()
    { return T; };

    // both the constructor and setValue can throw an
    // IntegerConstraintError
};
```

Temperature Controller cont.

```
class ActuatorDead extends Exception
{
    public String getMessage()
    { return ("Actuator Dead"); }
};

class TemperatureController
{
    public TemperatureController(int T)
        throws IntegerConstraintError
    {
        currentTemperature = new Temperature(T);
    };

    Temperature currentTemperature;

    public void setTemperature(int T)
        throws ActuatorDead, IntegerConstraintError
    { currentTemperature.setValue(T); };

    int readTemperature()
    { return currentTemperature.readValue(); }
};
```


Java: Exception Declaration



- Each function must specify a list of throwable checked exceptions **throw A, B, C**
 - Function may throw any exception in this list and any of the unchecked exceptions.
- A, B and C must be subclasses of `Exception`
- If a function attempts to throw an exception which is not allowed by its throws list:
 - *Compilation error* occurs

Temperature Controller cont.



```
import exceptionLibrary.IntegerConstraintError;
class Temperature
{
    int T;

    void check(int value) throws IntegerConstraintError
    {
        if(value > 100 || value < 0) {
            throw new IntegerConstraintError(0, 100, value);
        }
    }

    public Temperature(int initial) throws IntegerConstraintError
        // constructor
    { check(initial); T = initial; }

    public void setValue(int V) throws IntegerConstraintError
    { check(V); T = V; };

    public int readValue()
    { return T; };
};
```

Temperature Controller cont.



```
// given TemperatureController

try {
    TemperatureController TC = new TemperatureController(20);

    TC.setTemperature(100);
    // Statements which manipulate the temperature
}
catch (IntegerConstraintError error) {
    // Exception caught, print error message on
    // the standard output
    System.out.println(error.getMessage());
}
catch (ActuatorDead error) {
    System.out.println(error.getMessage());
}
```

The catch Statement



- The **catch** statement is like a function declaration
 - Parameter identifies the exception type to be caught
 - Inside the handler, the object name behaves like a local variable
- A handler with parameter type T will catch a thrown object of type E if:
 - T and E are the *same type*, or
 - T is a *parent (super) class* of E at the throw point
- This last point makes the Java exception handling facility very powerful

Java: Catching All



```
try {  
    // statements which might raise the exception  
    // IntegerConstraintError or ActuatorDead  
}  
catch(Exception E) {  
    // Handler will catch all exceptions of  
    //     type exception and any derived type;  
    // However, from within the handler only the  
    //     methods of Exception are accessible  
}
```

- A call to `E.getMessage` will dispatch to the appropriate routine for the type of object thrown
- **`catch(Exception E)`** is equivalent to Ada's **`when others`**

- Java supports a **finally** clause as part of a try statement
- Code here is guaranteed to execute whatever happens in the try statement irrespective of whether exceptions are thrown, caught, propagated or, indeed, even if there are no exceptions thrown at all

```
try
{
    ...
}
catch(...)
{
    ...
}
finally
{
    // code executed
    // under all
    // circumstances
}
```



- C does not define any exception handling facilities
- *This clearly limits the usefulness of C for the structured programming of reliable systems*
- However, it is possible to provide some form of exception handling mechanism by using the C macro facility
- To implement a termination model, it is necessary to save the status of a program's registers etc. on entry to an exception domain and then restore them if an exception occurs

C: Setjmp and Longjmp



- The *POSIX* facilities **setjmp** and **longjmp** can be used to implement a termination model
- **setjmp** *saves the program status* and returns a 0
- **longjmp** *restores the program status* and results in the program abandoning its current execution and restarting from the position where **setjmp** was called
 - This time **setjmp** returns the values passed by **longjmp**

Example with Setjmp and Longjmp




```
/* begin exception domain */
typedef char *exception; /* Pointer type to character string */
exception error = "error"; /* Represents exception "error" */

if ((current_exception = (exception) setjmp(save_area)) == 0) {
    /* save the registers and so on in save_area, returned 0 */

    /* the guarded region */
    /* when an exception "error" is identified */
    longjmp(save_area, (int) error);
    /* no return */
}
else {
    if (current_exception == error) {
        /* handler for "error" */
    }
    else {
        /* re-raise exception in surrounding domain */
    }
}
```

C Macros for Exception Handling



- Evaluation of plain `set jmp` and `long jmp` regarding R1 (Simplicity) and R2 (Unobtrusiveness):

- The use of C macros can help here

```
#define NEW_EXCEPTION(name) ...  
    /* code for declaring an exception */  
#define BEGIN ...  
    /* code for entering an exception domain */  
#define EXCEPTION ...  
    /* code for beginning exception handlers */  
#define END ...  
    /* code for leaving an exception domain */  
#define RAISE(name) ...  
    /* code for raising an exception */  
#define WHEN(name) ...  
    /* code for handler */  
#define OTHERS ...  
    /* code for catch all exception handler */
```

A Termination Model using C Macros



```
NEW_EXCEPTION(sensor_high);
NEW_EXCEPTION(sensor_low);
NEW_EXCEPTION(sensor_dead);
/* other declarations */

BEGIN
  /* statements which may cause the above */
  /* exceptions to be raised; for example */
  RAISE(sensor_high);

EXCEPTION
  WHEN(sensor_high)
    /* take some corrective action */
  WHEN(sensor_low)
    /* take some corrective action */
  WHEN(OTHERS)
    /* sound an alarm */
END;
```

Recovery Blocks and Exceptions



- Remember:

```
ensure  <acceptance test>
by
    <primary module>
else by
    <alternative module>
else by
    <alternative module>
    ...
else by
    <alternative module>
else error
```

- Error detection is provided by the acceptance test
 - The negation of a test which would raise an exception
- The only problem is the implementation of state saving and state restoration

A Recovery Cache



- Consider:

```
package Recovery_Cache is  
  procedure Save; -- save volatile state  
  procedure Restore; --restore state  
end Recovery_Cache;
```

- Body may require support
 - from the run-time system
 - possibly even from hardware
- Also, may be ineffective for state restoration
 - May be more desirable to provide more basic primitives
 - Would allow program to use its knowledge of the application to optimise the amount of information saved

Recovery Blocks in Ada



```
procedure Recovery_Block is
  Primary_Failure, Secondary_Failure,
  Tertiary_Failure: exception;
  Recovery_Block_Failure : exception;
type Module is (Primary, Secondary, Tertiary);

function Acceptance_Test return Boolean is
begin
  -- code for acceptance test
end Acceptance_Test;
```

Recovery Blocks in Ada cont.



```
procedure Primary is
begin
  -- code for primary algorithm
  if not Acceptance_Test then
    raise Primary_Failure;
  end if;
exception
  when Primary_Failure =>
    -- forward recovery to return environment
    -- to the required state
    raise;
  when others =>
    -- unexpected error
    -- forward recovery to return environment
    -- to the required state
    raise Primary_Failure;
end Primary;
-- similarly for Secondary and Tertiary
```

Recovery Blocks in Ada cont.



```
begin
  Recovery_Cache.Save;
  for Try in Module loop
    begin
      case Try is
        when Primary => Primary; exit;
        when Secondary => Secondary; exit;
        when Tertiary => Tertiary;
      end case;
    exception
      when Primary_Failure =>
        Recovery_Cache.Restore;
      when Secondary_Failure =>
        Recovery_Cache.Restore;
      when Tertiary_Failure =>
        Recovery_Cache.Restore;
        raise Recovery_Block_Failure;
      when others =>
        Recovery_Cache.Restore;
        raise Recovery_Block_Failure;
      end;
    end loop;
  end Recovery_Block;
```


Exceptions and Timing Faults



- **Recall:** It is necessary to be able to detect
 - overrun of deadline
 - overrun of worst-case execution time
 - sporadic events occurring more often than predicted
 - timeout on communications
- Exceptions may be used to indicate such timing faults
- May use exceptions to trigger an **event-based reconfiguration** (performing a **mode change**)
 - Alter process deadlines or suspend/terminate processes
 - Start new processes
 - Ask a process to immediately return best result obtained so far

Exceptions in Real-Time Euclid



- Real-Time Euclid combines asynchronous event handling with its real-time abstractions
 - Time constraints are associated with processes
 - Can define numbered exceptions
- A process may raise an exception in another process
- Three kinds of raise statement:
 - **except** – similar to Ada raise, but with resumption
 - **deactivate** – terminates current iteration of (periodic) process
 - **kill** – explicitly removes a process from list of active processes (but still executes exception handler)

Real-Time Euclid: Example



```
process TempController: periodic frame 60
  first activation atTime 600 or atEvent startMonitoring

  handler (except_num)
    exceptions (200, 201, 304)
    imports (var consul, ...)
    var message: string(80), ...
    case except_num of
      label 200: % very low temperature
        message := "reactor is shut down"
        consul := message
      label 201: % very high temperature
        message := "reactor is shut down"
        consul := message
        alarm := true      % activate alarm device
      label 304: % timeout on sensor
        % reboot sensor device
    end case
  end handler
```

Real-Time Euclid: Example cont.



```
...

wait(temp_available) noLongerThan 10: 304

currentTemp := ...    % Low-level i/o
log := currentTemp

if currentTemp < 100 then
    deactivate TempController: 200

elseif currentTemp > 10000 then
    kill TempController: 201

end if

% Other computations

end TempController
```



- It is not unanimously accepted that exception handling facilities should be provided in a language
- The occurrence of an exception often requires a two-fold action
 - an *internal clean-up*, to ensure *failure atomicity*
 - an *external error treatment*
- Exceptions may implement recovery blocks
 - Also allows to perform forward error recovery before restoring the state
- Exceptions may also handle timing faults
 - Facilitates dynamic reconfiguration

- For example, C and occam2 have no exceptions
- *To sceptics, an exception is a GOTO where the destination is undeterminable and the source is unknown!*
- Exceptions can, therefore, be considered to be the antithesis of structured programming
- However, this is not the view taken here!

Summary II



Language	Domain	Propagation	Model	Parameters
Ada	Block	Yes	Termination	Limited
Java	Block	Yes	Termination	Yes
C++	Block	Yes	Termination	Yes
CHILL	Statement	No	Termination	No
CLU	Statement	No	Termination	Yes
Mesa	Block	yes	Hybrid	Yes

- C supports a low-level exception handling mechanism using `set jmp` and `long jmp`
- No language gets perfect scores in all five criteria (simplicity, unobtrusiveness, efficiency, uniformity, recovery)

Problem Set 8 – Due: (Mon) 17 June 2002



Modify the robot built last week such that

- a) In case it drives for *more than 1m* without finding a dark line, it stops and raises an acoustic alarm
- b) In case it finds a line and this line is *thicker than 10cm*, it also stops, and raises another type of alarm

You should provide two versions of the controller, both using *exceptions* to implement the two cases described above.

- 1) In C – using the C exception-handling macros introduced in class (yes, defining these macros is part of the problem :-)
- 2) In Java – using for example lejos (see <http://www.informatik.uni-kiel.de/~kwi/programmierung/lejos.html>)

a) *Documentation* (overview of approach and assumptions, commented source code, measurement of accuracy)

(2x3 pts)

b) Functional *robot* **(2x3 pts + up to 3 bonus points for best entry)**

Enjoy!