

# Exception handling

Amer Diwan

## What is exception handling

- A civilized way of dealing with exceptional situations
  - When a code detects an error, it raises an exception
  - Code that knows how to handle the exception can declare “handlers” for the exception

## An example in Modula-3

```
EXCEPTION BadValueError
BEGIN
  TRY
    IF i < 0 THEN
      RAISE BadValueError;
    ELSE
      i = anotherI();
    END;
  EXCEPT
    BadValueError => Print("Ouch\n");
  END;
END
```

## How do we find the handler?

- Modula-3, Java, C++: Static Scoping
- PL/1: Dynamic scoping

## Example of static scoping in Modula-3

```
EXCEPTION BadValueError1, BadValueError2;  
BEGIN  
  TRY  
    ...  
  EXCEPT  
    BadValueError1 => Print("OuchB\n");  
  END;  
  TRY  
    ... RAISE BadValueError1; ...  
  EXCEPT  
    BadValueError1 => Print("OuchA\n");  
  END;  
END
```

## Another example of static scoping

```
EXCEPTION BadValueError1, BadValueError2;  
BEGIN  
  TRY  
    TRY  
      ... RAISE BadValueError1; ...  
    EXCEPT  
      BadValueError1 => Print("OuchA\n");  
    END;  
  EXCEPT  
    BadValueError1 => Print("OuchB\n");  
  END;  
END
```

## Yet another example of static scoping

```
EXCEPTION BadValueError1, BadValueError2;
BEGIN
  TRY
    TRY
      ... RAISE BadValueError2; ...
    EXCEPT
      BadValueError1 => Print("OuchA\n");
    END;
  EXCEPT
    BadValueError2 => Print("OuchB\n");
  END;
END
```

## An example of dynamic scoping

- ON CONDITION explosion PRINT “nice knowing you”  
BEGIN  
IF b is true THEN  
ON CONDITION explosion PRINT “gotcha”  
SIGNAL explosion;  
END  
SIGNAL explosion
- Advantages: very powerful
- Disadvantages: hard to understand

## How is an exception raised?

- Can use a RAISE/throw statement
- Language can automatically raise an exception

```
TRY
  ... *p ...
EXCEPT
  NullPointerException => Print("Ouch\n");
END;
END;
```

- Advantage of implicit: uniform treatment of errors, but...

## What happens after a handler executes

- **Termination model** (Java, Modula-3, C++...)
  - Execution continues after the handler
- **Resumption model** (PL/1)
  - Execution continues after the RAISE
  - Complex
  - Perhaps more directly supports error recovery

## Example of termination model

```
EXCEPTION BadValueError
BEGIN
  TRY
    ...RAISE BadValueError;...
    Print("Ouch2\n");
  EXCEPT
    BadValueError => Print("Ouch1\n");
  END;
  Print("Ouch3\n");
END
```

## Example of resumption model

- **ON CONDITION** explosion **PRINT** “nice knowing you”  
**BEGIN**  
  **IF** b is true **THEN**  
    **ON CONDITION** explosion **PRINT** “gotcha”  
    **SIGNAL** explosion;  
  **END**  
**SIGNAL** explosion

## What happens when...

(Most of the following discussion applies to static scoping of handlers and the termination model)

A routine returns without handling the exception?

- The routine returns, and
- The exception is re-raised at the call to the routine

## An example

```
PROCEDURE f() = BEGIN
  ... RAISE BadValueError;
  Print("Ouch4\n");
END f;
BEGIN
  TRY
    f();
    Print("Ouch2\n");
  EXCEPT BadValueError => Print("Ouch1\n"); END;
  Print("Ouch3\n");
END
```

## Continuing with example

```
PROCEDURE f() = BEGIN
  TRY
    ... RAISE BadValueError;
  EXCEPT DivZero => ... END;
  Print("Ouch4\n");
END f;
BEGIN
  TRY
    f();
    Print("Ouch2\n");
  EXCEPT BadValueError => Print("Ouch1\n");END;
  Print("Ouch3\n");
END
```

## Another example

- PROCEDURE f() =  
 BEGIN ... RAISE BadValue; ... END f;
- PROCEDURE g() =  
 BEGIN ... f(); END g;
- BEGIN  
 TRY  
 f();  
 EXCEPT BadValue => ... END;  
 END Main.
- What if Main doesn't handle the exception?

## A problem with propagation of exceptions

- ```
PROCEDURE f() =  
  TRY  
    h();  
  EXCEPT  
    ???  
END;
```
- What exceptions does "h" raise? What exceptions does "f" (or its callers) need to handle?

## Approach taken by Modula-3/Java/...

- The declaration of each procedure says what exceptions it can raise
- ```
PROCEDURE f() RAISES {BadValueExc} = ...
```
- ```
class BadValueExc extends Exception { ... }  
void f() throws BadValueExc { ... }
```

## Example

- ```
void f() {  
    ...throw BadValueExc; ...  
}
```
- Is this legal?

## Another example

- ```
void f() throws BadValueExc {  
    ...throw BadValueExc; ...  
}
```
- ```
void g() {  
    f();  
}
```

## Problem with "throws" clause

- In Java, all errors, such as "div 0" map to exceptions
- ```
void f() {  
    ...p.next...x/y...a[i]  
}
```
- "f" must have a handler for every exception that may be raised in it or mention it in the throws clause

## Two ways of handling it

- ```
void f() throws NullPointerException, Div0, ... {  
    ...p.next...x/y...a[i]...  
}
```
- ```
void f() throws Exception {  
    ...p.next...x/y...a[i] }
```
- What are the problems with each of these approaches?

## Java's solution: checked versus unchecked exceptions

- The language guarantees at compile time that either
  - there is an enclosing handler for every throw of a **checked** exception, or
  - the enclosing procedure names all potentially raised **checked** expressions (or their supertypes) in its “throws” clause
- A program does not need handlers for **unchecked** exceptions: all routines can raise them

## Form of an exception

- In Java exceptions are objects, in Modula-3 they are a special type
- ```
class E1 extends Exception {...}
class E2 extends E1 {...}
class E3 extends E1 {...}
try {
    throw one of E1, E2, E3
}
catch (E1 e1) {...}
```

## Another example

- ```
class E1 extends Exception {...}
class E2 extends E1 {...}
class E3 extends E1 {...}
try {
    throw one of E1, E2, E3
}
catch (E2 e2) {...}
catch (E1 e1) {...}
```
- In Modula-3, exceptions are not organized with subclassing

## Exceptions as classes versus exceptions as a special type

- Expressiveness
- Simplicity
- Safety
- Ease of implementation
- Efficiency

## Exceptions versus return codes for error handling

- Expressiveness
- Simplicity
- Safety
- Ease of implementation
- Efficiency

## Relationship to reading

- Reading goes into more detail on PL/1 and Ada's model.
- Reading gives some details on how to implement exceptions