Exception handling

Exceptions

- Exceptions = a mechanism for dealing with run-time occurrences of errors (such as improperly formed input data), and whose occurrence causes an unconditional transfer of control
- in Ada is an object whose "type" is Exception
- an exception in Ada can be raised, and it can be handled; information associated with an occurrence of an exception can be handled

Exceptions

- Ada uses heavily exceptions especially for data consistency
- check failures at run time
- checking against type ranges and array boundaries,
- null pointers,
- Various kind of concurrency properties
- functions not returning a value

```
procedure Example is
X: Natural;
Begin
X:=-100;
end Example;
```

Exceptions

- the code will raise Constraint_Error since we assign negative value to a natural variable.
- the compiler may give a warning that the value is out of range, but the error manifests as a run-time exception.
- there is no local handler, the exception is propagated to the caller;
- if is the main procedure, then the program will be terminated

Exceptions

- In Ada all the code is implicitly considered as a try-block, and in case of errors exceptions are raised automatically
- Unlike other languages, Ada uses *raising*, not *throwing* an exception, and *handling*, not *catching*, an exception.

```
package Exceptions is
My_Except : exception;   -- Like an object. *NOT* a type
end Exceptions;
```

- to raise an exception of our newly declared exception kind, do the following:

Main

```
with Exceptions; use Exceptions;
procedure Main is
begin
raise My_Except;
--  Execution of current control flow abandoned; an exception
of kind
--  "My_Except" will bubble up until it is caught.
raise My_Except with "My exception message";
--  Execution of current control flow abandoned; an exception
of   kind "My_Except" with associated string will bubble up
until   it is caught.
end Main;
```

Exceptions

- handling is done at the end of programming units, e.g.

```
begin
 -- statements;
exception
    when My_Except  =>  Put_Line ( „handling errors” );
end;
```

- in handler we usually just print a message, large code is never written here

Exceptions

- Propagation of the errors is done in dynamic order, i.e the programs and subprograms are calling each other (an not the static order, the way the subprograms are embedded in each other).

If the error appears in the main program, then the exception handling is searched in the main.

If in any subprograms, then first in the subprogram, if does not exist, then in its caller, if no handler, then the next caller, up to the main program.

If no handler found in the dynamic chain (calling order), then the main will run with the error.

Types of Exceptions

- Standard Exceptions . 4
- Custom Exceptions
- An exception in Ada is an object whose "type" is exception, as opposed to classes in Java or any type in C++.
- Basically, an exception in Ada can be raised, and it can be handled; information associated with an occurrence of an exception can be interrogated by a handler.

Predefined exceptions

- Constraint_Error - raised when a value goes out of the type's legal range.
- Program_Error - raised when control rules are broken, for example when running into the **end** of a function before a **return** statement is encountered.
- Storage_Error - raised when an allocator requests storage and not enough is available.
- Tasking_Error - raised when some improper task interaction occurs, such as when a called task is completed before a queued entry call is accepted.

Custom Exceptions

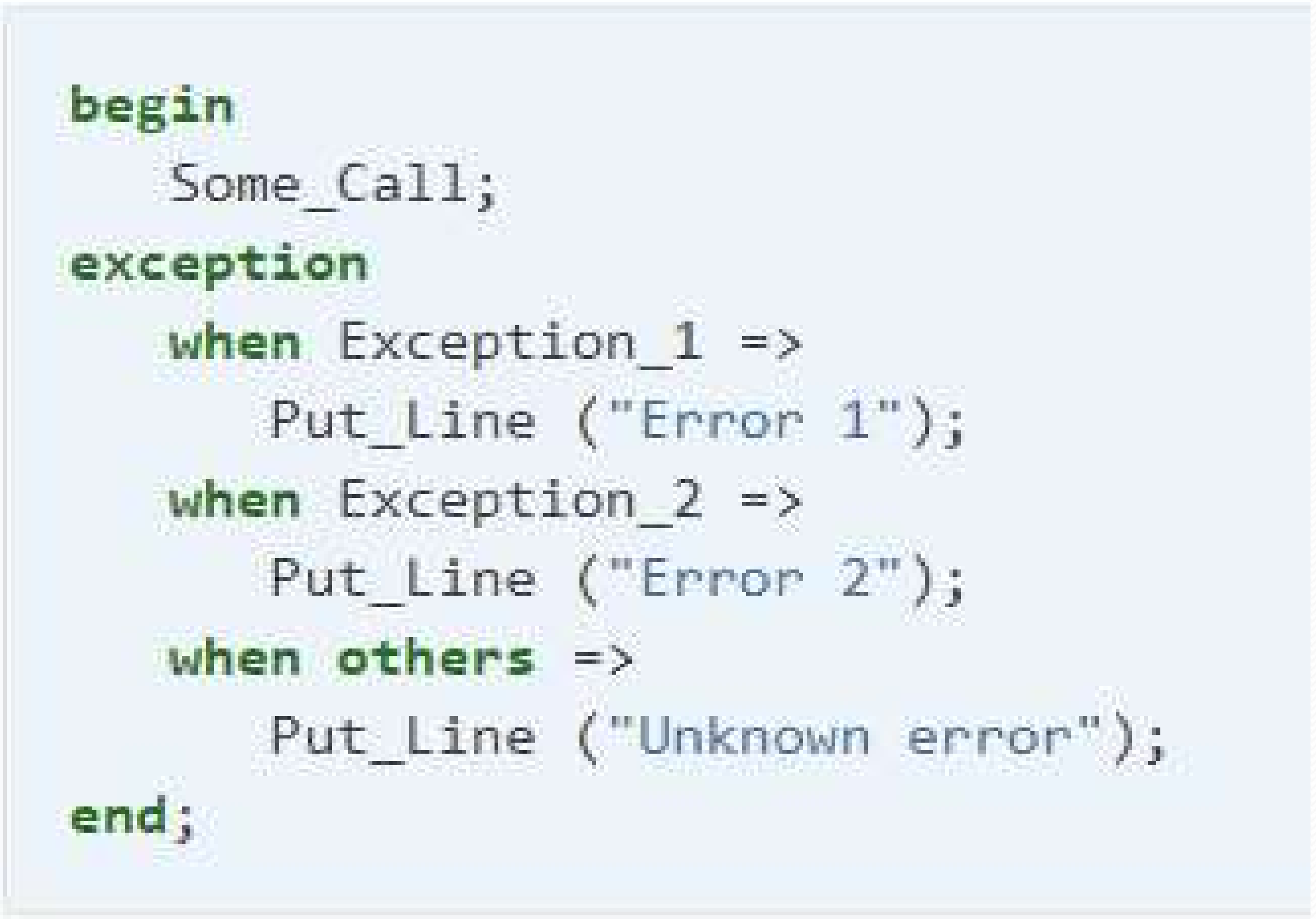
- Custom exception declarations resemble object declarations, and they can be created in Ada using the *exception* keyword:

```
My_Exception : exception;
```

It can be raised in the program:

```
raise My_Exception with “some message” ;
```

Exception handling



```
begin
   Some_Call;
exception
   when Exception_1 =>
      Put_Line ("Error 1");
   when Exception_2 =>
      Put_Line ("Error 2");
   when others =>
      Put_Line ("Unknown error");
end;
```