Programming Paradigms

Control Abstraction (Part 3)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- Calling Sequences
- Parameter Passing
- Exception Handling 
- Coroutines
- Events

Warm-up Exercise

What does the following Java code print?

```
try {
    try {
        Object obj = null;
        obj.equals(obj);
    } catch (IllegalStateException e) {
        System.out.println("Caught it.");
    } catch (NullPointerException e) {
        throw new RuntimeException(e);
    }
} catch (NullPointerException e) {
    System.out.println("Caught it, too.");
} finally {
    System.out.println("Finally here.");
}
```

Please vote in Ilias.

Warm-up Exercise

What does the following Java code print?

```
try {
    try {
        Object obj = null;
        obj.equals(obj);
    } catch (IllegalStateException e) {
        System.out.println("Caught it.");
    } catch (NullPointerException e) {
        throw new RuntimeException(e);
    }
} catch (NullPointerException e) {
    System.out.println("Caught it, too.");
} finally {
    System.out.println("Finally here.");
}
```

Result:

Finally here.

Exception in ...

Please vote in Ilias.

Warm-up Exercise

What does the following Java code print?

```
try {
    try {
        Object obj = null;
        obj.equals(obj);
    } catch (IllegalStateException e) {
        System.out.println("Caught it.");
    } catch (NullPointerException e) {
        throw new RuntimeException(e);
    }
} catch (NullPointerException e) {
    System.out.println("Caught it, too.");
} finally {
    System.out.println("Finally here.");
}
```

Please vote in Ilias.

Warm-up Exercise

What does the following Java code print?

```
try {
    try {
        Object obj = null;
        obj.equals(obj);
    } catch (IllegalStateException e) {
        System.out.println("Caught it.");
    } catch (NullPointerException e) {
        throw new RuntimeException(e);
    }
} catch (NullPointerException e) {
    System.out.println("Caught it, too.");
} finally {
    System.out.println("Finally here.");
}
```

Wrong exception type:

Nothing caught here.

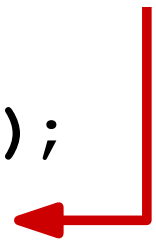
Please vote in Ilias.

Warm-up Exercise

What does the following Java code print?

```
try {
    try {
        Object obj = null;
        obj.equals(obj);
    } catch (IllegalStateException e) {
        System.out.println("Caught it.");
    } catch (NullPointerException e) {
        throw new RuntimeException(e);
    }
} catch (NullPointerException e) {
    System.out.println("Caught it, too.");
} finally {
    System.out.println("Finally here.");
}
```

Catches `e` and wraps it into another exception



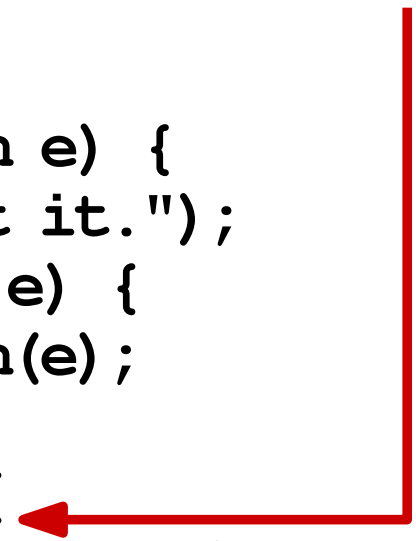
Please vote in Ilias.

Warm-up Exercise

What does the following Java code print?

Not a `NullPointerException` anymore: Nothing caught here

```
try {
    try {
        Object obj = null;
        obj.equals(obj);
    } catch (IllegalStateException e) {
        System.out.println("Caught it.");
    } catch (NullPointerException e) {
        throw new RuntimeException(e);
    }
} catch (NullPointerException e) {
    System.out.println("Caught it, too.");
} finally {
    System.out.println("Finally here.");
}
```



Please vote in Ilias.

Warm-up Exercise

What does the following Java code print?

```
try {
    try {
        Object obj = null;
        obj.equals(obj);
    } catch (IllegalStateException e) {
        System.out.println("Caught it.");
    } catch (NullPointerException e) {
        throw new RuntimeException(e);
    }
} catch (NullPointerException e) {
    System.out.println("Caught it, too.");
} finally {
    System.out.println("Finally here.");
}
```

`finally` blocks are always executed

Please vote in Ilias.

Exceptions

- **Exception: Unusual condition during execution that cannot be easily handled in local context**
- **Raising an exception diverges from normal control flow**
- **Exception handler: Code executed when an exception occurs**

When Do Exceptions Occur?

- **Implicitly thrown by language implementation**
  - Runtime errors, e.g., division by zero
- **Explicitly thrown by program**
  - Illegal or unexpected program state, e.g., combination of flags that should never occur
- **Don't use exceptions to encode "normal" control flow**

Alternatives to Exceptions

In PL without exceptions, three other options

- "Invent" a return value
  - E.g., empty string if cannot read from file
- Encode status in return value
  - E.g., as an integer error code
- Caller passes a closure with error-handling routine
  - E.g., "error-first" callback on Node.js

Syntax of Exceptions

Most common in modern PLs:

Try-catch blocks

- Handler is lexically bound to block of code
- Example (C++):

```
try {
  // ...
  if (something_unexpected)
    throw my_error("oops");
  // ...
} catch (my_error e) {
  // handle exception
}
```

Syntax of Exceptions

Most common in modern PLs:

Try-catch blocks

- Handler is lexically bound to block of code
- Example (C++):

```
try {
  // ...
  if (something_unexpected)
    throw my_error("oops");
  // ...
} catch (my_error e) {
  // handle exception
}
```

Handler for specific type of exception

Nested Try Blocks

- If exception thrown, control passed to inner-most matching handler

```
try {
  try {
    // ...
    // code that may throw exception
    // ...
  } catch (some_other_error e) {
    // handle some_other_error
  }
} catch (my_error e) {
  // handle my_error
}
```

Nested Try Blocks

- If exception thrown, control passed to inner-most matching handler

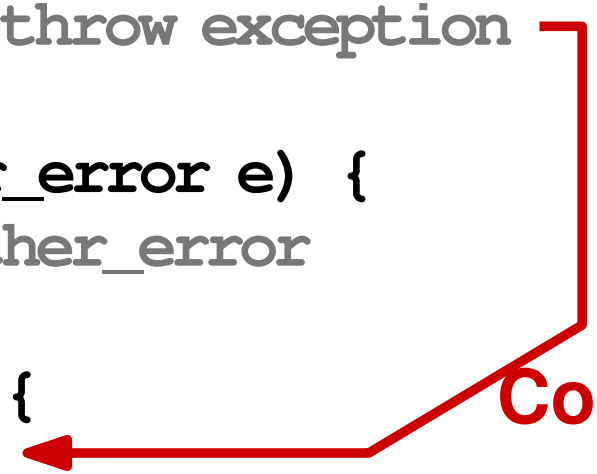
```
try {
  try {
    // ...
    // code that may throw exception
    // ...
  } catch (some_other_error e) {
    // handle some_other_error
  }
} catch (my_error e) {
  // handle my_error
}
```

Control flow if `some_other_error` thrown

Nested Try Blocks

- If exception thrown, control passed to inner-most matching handler

```
try {
  try {
    // ...
    // code that may throw exception
    // ...
  } catch (some_other_error e) {
    // handle some_other_error
  }
} catch (my_error e) {
  // handle my_error
}
```



Control flow if `my_error` thrown

Lists of Handlers

- If different exceptions thrown in same block, use list of handlers

```
try {
  // code that may throw exception
} catch (end_of_file e) {
  // handle end of file
} catch (io_error  e) {
  // handle I/O errors
} catch (...) {
  // handles any not previously handled exception
}
```

Lists of Handlers

- If different exceptions thrown in same block, use list of handlers

```
try {
  // code that may throw exception
} catch (end_of_file e) {
  // handle end of file
} catch (io_error  e) {
  // handle I/O errors
} catch (...) {
  // handles any not previously handled exception
}
```

C++ syntax for "catch all"

Propagation Outside Subroutine

What if no matching handler in current subroutine?

- Immediately return and re-raise exception at call site
- May propagate until main routine
  - Unwinds stack without finishing routines
- If not handled at all, terminate program

Defining Exceptions

Mechanisms vary across PLs

- Subtype of particular class
  - E.g., in Java, subtypes of `Exception`
- Special kinds of objects (akin to constants, types, variables)
  - E.g., in Modula-3:

    ```
    EXCEPTION empty_queue
    ```
- Any value that exists in the PL
  - E.g., JavaScript:

    `throw 42;` or `throw "Expected a number";`

How to Handle Exceptions?

- **Recover and continue execution**
  - E.g., if out of memory, allocate more memory
- **Clean up locally allocated resources and re-raise exception to handled elsewhere**
  - E.g., close opened files
- **Print error message and terminate program**

How to Handle Exceptions?

- **Recover and continue execution**
  - E.g., if out of memory, allocate more memory
- **Clean up locally allocated resources and re-raise exception to handled elsewhere**
  - E.g., close opened files
- **Print error message and terminate program**

Do not just swallow exceptions!

Declaring Exceptions

In some PLs, possibly thrown exceptions are part of the subroutine header

- Must declare every exception, e.g., Modula-3
- Declaring exceptions is optional, e.g., C++
- Checked vs. unchecked exceptions, e.g., Java
  - Must declare checked exceptions
  - Optional for unchecked exceptions

Cleanup Operations

- **`finally` clause: Executed whenever control leaves the current block**
  - When exception is thrown
  - Also when no exception thrown
- **Use to clean up local state**
  - E.g., release resources

Quiz: Exceptions

What does this Python code print?

```
def f():
    try:
        print("a")
    except:
        print("b")
    finally:
        g()
        print("c")

def g():
    try:
        raise "oops"
    except:
        print("d")
    finally:
        print("e")

f()
```

Please vote in Ilias.

Quiz: Exceptions

What does this Python code print?

Result:

a

d

e

c

```
def f():
    try:
        print("a")
    except:
        print("b")
    finally:
        g()
        print("c")

def g():
    try:
        raise "oops"
    except:
        print("d")
    finally:
        print("e")

f()
```

Please vote in Ilias.