

14 Handling errors

Main concepts to be covered

- Defensive programming.
 - Anticipating that things could go wrong.
- Exception handling and throwing.
- Error reporting.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 2

Some causes of error situations

- Incorrect implementation.
 - Does not meet the specification.
 - Inappropriate object request.
 - E.g., invalid index.
 - Inconsistent or inappropriate object state.
 - E.g. arising through class extension.
- example in slides 4-11 →*

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 3

Example: Class extension

```
public class Point {  
    private int x,y;  
    public Point(int x,int y) { this.x = x; this.y = y; }  
    public int getX() { return x; }  
    public int getY() { return y; }  
}
```

```
public class Rectangle extends Point {  
    private int width,height;  
    public Rectangle(int x,int y,int width,int height) {  
        super(x,y);  
        setWidth(width); setHeight(height);  
    }  
    public int getWidth() { return width; }  
    public int getHeight() { return height; }  
    public void setWidth(int width) { this.width = width; }  
    public void setHeight(int height) { this.height = height; }  
}
```

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 4

Class extension (2)

- Want more subclasses ...
- A **square** is a rectangle with equal sides.
- A **circle** is an ellipse with equal axis.
- Idéa:
 - Define Square as a subclass to Rectangle,
 - Circle as a subclass to Ellipse, etc.
- *Is this idéa good or bad?*

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 5

A Square IS A Rectangle (?)

```
Rectangle  
|  
+-- public class Square extends Rectangle {  
|     public Square(int x,int y,int size) {  
|         super(x,y,size,size);  
|     }  
|     public int getSize() { return getWidth(); }  
|     public void setSize(int size) {  
|         setWidth(size);  
|         setHeight(size);  
|     }  
|     public int getArea() {  
|         return getSize()*getSize();  
|     }  
| }
```

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 6

Class extension (4)

```
Square sq = new Square(12,34,100);
System.out.println(sq.getSize()); // 100    OK!
sq.setWidth(200);
System.out.println(sq.getSize()); // 200    OK!
sq.setHeight(300);
System.out.println(sq.getSize()); // 200    OK!...?
System.out.println(sq.getArea()); // 40000  OK!
System.out.println(sq.getWidth()*
                        sq.getHeight()); // 60000 #@*!
```

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 7

Class extension (5)

- If a Square really IS A Rectangle, then the following equalities should hold for any Square s:
 - `s.getWidth() == s.getHeight()`
 - `s.getWidth() == s.getSize()`
 - `s.getHeight() == s.getSize()`
- *Square has to override Rectangle methods in order to enforce those equalities!*

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 8

Class extension (6) Override critical Rectangle methods

```
public class Square extends Rectangle {
    public Square(int x,int y,int size) {
        super(x,y,size,size);
    }
    public int getSize() { return getWidth(); }
    public void setSize(int size) {
        super.setWidth(size);
        super.setHeight(size);
    }
    public int getArea() { return getSize()*getSize(); }
    @Override
    public int getHeight() { return getSize(); }
    @Override
    public void setHeight(int height) { setSize(height); }
    @Override
    public void setWidth(int width) { setSize(width); }
}
```

- *Does this feel as a clean extension?*

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 9

Class extension (7)

- Concl. A Square IS NOT A Rectangle
 - because a Rectangle cannot be guaranteed to behave (conceptually) as a square.
- The basic philosophy is that inheritance should extend a class with additional properties, not restrict a class from having certain properties.
- *So this extension was a bad idéa ...*

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 10

Alternative solution: Aggregation - A Square HAS A Rectangle

```
public class Square {
    private Rectangle rect;
    public Square(int x,int y,int size) {
        rect = new Rectangle(x,y,size,size);
    }
    public int getSize() { return rect.getWidth(); }
    public void setSize(int size) {
        rect.setWidth(size);
    }
    public int getArea() {
        rect.getWidth();
    }
}
```

- *Square uses a Rectangle for implementation purposes.*

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 11

Not always programmer error

- Errors often arise from the environment:
 - Incorrect URL entered.
 - Network interruption.
- File processing is particular error-prone:
 - Missing files.
 - Lack of appropriate permissions.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 12

Exploring errors

- Explore error situations through the *address-book* projects.
- Two aspects:
 - Error reporting.
 - Error handling.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 13

Defensive programming

- Client-server interaction.
 - Should a server assume that clients are well-behaved?
 - Or should it assume that clients are potentially hostile?
- Significant differences in implementation required.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 14

Issues to be addressed

- How much checking by a server on method calls?
- How to report errors?
- How can a client anticipate failure?
- How should a client deal with failure?

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 15

An example

- Create an AddressBook object.
- Try to remove an entry.
- A runtime error results.
 - Whose 'fault' is this?
- Anticipation and prevention are preferable to apportioning blame.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 16

Argument values

- Arguments represent a major 'vulnerability' for a server object.
 - Constructor arguments initialize state.
 - Method arguments often contribute to behavior.
- Argument checking is one defensive measure.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 17

Checking the key

```
public void removeDetails(String key)
{
    if (keyInUse(key)) {
        ContactDetails details = book.get(key);
        book.remove(details.getName());
        book.remove(details.getPhone());
        numberOfEntries--;
    }
}
```

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 18

Server error reporting

- How to report illegal arguments?
 - To the user?
 - Is there a human user?
 - Can they solve the problem?
 - To the client object?
 - Return a diagnostic value.
 - *Throw an exception.*

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 19

Returning a diagnostic

```
public boolean removeDetails(String key)
{
    if (keyInUse(key)) {
        ContactDetails details = book.get(key);
        book.remove(details.getName());
        book.remove(details.getPhone());
        numberOfEntries--;
        return true;
    }
    else {
        return false;
    }
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 20

Client responses

- Test the return value.
 - Attempt recovery on error.
 - Avoid program failure.
- Ignore the return value.
 - Cannot be prevented.
 - Likely to lead to program failure.
- Exceptions are preferable.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 21

Exception-throwing principles

- A special language feature.
- No 'special' return value needed.
- Errors cannot be ignored in the client.
 - The normal flow-of-control is interrupted.
- Specific recovery actions are encouraged.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 22

Throwing an exception

```
/**
 * Look up a name or phone number and return the
 * corresponding contact details.
 * @param key The name or number to be looked up.
 * @return The details corresponding to the key,
 *         or null if there are none matching.
 * @throws NullPointerException if the key is null.
 */
public ContactDetails getDetails(String key)
{
    if (key == null) {
        throw new NullPointerException(
            "null key in getDetails");
    }
    return book.get(key);
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 23

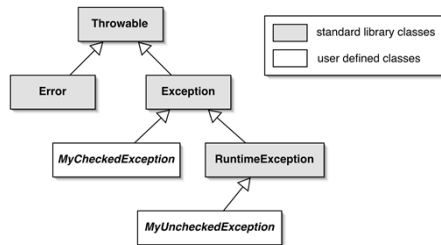
Throwing an exception

- An exception object is constructed:
 - `new ExceptionType("...");`
- The exception object is thrown:
 - `throw ...`
- Javadoc documentation:
 - `@throws ExceptionType reason`

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 24

The exception class hierarchy



Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 25

Exception categories

- Checked exceptions
 - Subclass of `Exception`
 - Use for anticipated failures.
 - Where recovery may be possible.
- Unchecked exceptions
 - Subclass of `RuntimeException`
 - Use for unanticipated failures.
 - Where recovery is unlikely.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 26

The effect of an exception

- The throwing method finishes prematurely.
- No return value is returned.
- Control does not return to the client's point of call.
 - So the client cannot carry on regardless.
- A client may 'catch' an exception.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 27

Unchecked exceptions

- Use of these is 'unchecked' by the compiler.
- Cause program termination if not caught.
 - This is the normal practice.
- `IllegalArgumentException` is a typical example.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 28

Argument checking

```
public ContactDetails getDetails(String key)
{
    if(key == null) {
        throw new NullPointerException(
            "null key in getDetails");
    }
    if(key.trim().length() == 0) {
        throw new IllegalArgumentException(
            "Empty key passed to getDetails");
    }
    return book.get(key);
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 29

Preventing object creation

```
public ContactDetails(String name, String phone, String address)
{
    if(name == null) {
        name = "";
    }
    if(phone == null) {
        phone = "";
    }
    if(address == null) {
        address = "";
    }

    this.name = name.trim();
    this.phone = phone.trim();
    this.address = address.trim();

    if(this.name.length() == 0 && this.phone.length() == 0) {
        throw new IllegalStateException(
            "Either the name or phone must not be blank.");
    }
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 30

Exception handling

- Checked exceptions are meant to be caught.
- The compiler ensures that their use is tightly controlled.
 - In both server and client.
- Used properly, failures may be recoverable.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 31

The throws clause

- Methods throwing a checked exception must include a throws clause:

```
public void saveToFile(String destinationFile)
    throws IOException
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 32

The try statement

- Clients catching an exception must protect the call with a try statement:

```
try {
    Protect one or more statements here.
}
catch(Exception e) {
    Report and recover from the exception here.
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 33

The try statement

```
try {
    addressbook.saveToFile(filename);
    tryAgain = false;
}
catch(IOException e) {
    System.out.println("Unable to save to " + filename);
    tryAgain = true;
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 34

Catching multiple exceptions

```
try {
    ...
    ref.process();
    ...
}
catch(EOFException e) {
    // Take action on an end-of-file exception.
    ...
}
catch(FileNotFoundException e) {
    // Take action on a file-not-found exception.
    ...
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 35

Defining new exceptions

- Extend RuntimeException for an unchecked or Exception for a checked exception.
- Define new types to give better diagnostic information.
 - Include reporting and/or recovery information.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 36


```
public class NoMatchingDetailsException extends Exception
{
    private String key;

    public NoMatchingDetailsException(String key)
    {
        this.key = key;
    }

    public String getKey()
    {
        return key;
    }

    public String toString()
    {
        return "No details matching '" + key +
            "' were found.";
    }
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 37

Assertions

- Used for *internal* consistency checks.
 - E.g. object state following mutation.
- Used during development and normally removed in production version.
 - E.g. via a compile-time option.
- Java has an *assert statement*.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 38

Java Assertion Statement

- Two forms available:
 - `assert boolean-expression`
 - `assert boolean-expression : expression`
- The boolean-expression expresses something that should be true at this point.
- An `AssertionError` is thrown if the assertion is false.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 39

Assert Statement

```
public void removeDetails(String key)
{
    if (key == null) {
        throw new IllegalArgumentException("...");
    }
    if (keyInUse(key)) {
        ContactDetails details = book.get(key);
        book.remove(details.getName());
        book.remove(details.getPhone());
        numberOfEntries--;
    }
    assert !keyInUse(key);
    assert consistentSize() :
        "Inconsistent book size in removeDetails";
}
```

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 40

Guidelines for Assertions

- They are not an alternative to throwing exceptions.
- Use for internal checks.
- Remove from production code.
- Don't include normal functionality:
`// Incorrect use:`
`assert book.remove(name) != null;`

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 41

Error recovery

- Clients should take note of error notifications.
 - Check return values.
 - Don't 'ignore' exceptions.
- Include code to attempt recovery.
 - Will often require a loop.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 42

Attempting recovery

```
// Try to save the address book.
boolean successful = false;
int attempts = 0;
do {
    try {
        addressbook.saveToFile(filename);
        successful = true;
    }
    catch(IOException e) {
        System.out.println("Unable to save to " + filename);
        attempts++;
        if(attempts < MAX_ATTEMPTS) {
            filename = an alternative file name;
        }
    }
} while(!successful && attempts < MAX_ATTEMPTS);
if(!successful) {
    Report the problem and give up;
}
```

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 43

Error avoidance

- Clients can often use server query methods to avoid errors.
 - More robust clients mean servers can be more trusting.
 - Unchecked exceptions can be used.
 - Simplifies client logic.
- May increase client-server coupling.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 44

Avoiding an exception

```
// Use the correct method to put details
// in the address book.
if(book.keyInUse(details.getName()) ||
    book.keyInUse(details.getPhone()) {
    book.changeDetails(details);
}
else {
    book.addDetails(details);
}
```

The `addDetails` method could now throw an unchecked exception.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 45

Exceptions and overriding

Requirement of the substitution principle:

- It should always be safe to replace an object of a base class with an object of a sub class.
- In particular, exception handlers for base class method calls should work equally well for calls to overriding methods of the sub class.
- All checked exceptions thrown by calls to overriding methods must be type compatible with exceptions that could be thrown by calls to overridden methods in the base class.

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 46

Exceptions and overriding

Type compatibility rule for throws clauses:

- The set of exceptions A declared in the throws clause in an overriding method must be type compatible with the set of exceptions B declared in the throws clause in the overridden method.
- A is type compatible with B if for every exception type E_1 in A, there is an exception type E_2 in B such that E_1 is a subtype of E_2 .

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 47

Exceptions and overriding Example

```
public class E1 extends Exception {}
public class E2 extends E1 {}
public class E3 extends Exception {}
```

```
public class Base {
    public void f() throws E1 {}
    public void g() throws E1,E3 {}
    public void h() throws E2 {}
}
```

```
public class Sub extends Base {
    ... see the following slides
}
```

Object oriented programming, DAT042, DA02, 12/13, lp 1

Förel. 14 48

Exceptions and overriding Example

```
public class Sub extends Base {  
    public void f() {}  
    public void f() throws E1 {}  
    public void f() throws E2 {}  
    public void f() throws E1,E2 {}  
  
    public void f() throws E3 {}  
    public void f() throws E1,E3 {}  
    public void f() throws E2,E3 {}  
    ...  
}
```

Any of these is a
correct overring
of f

... but NONE
of these

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 49

Exceptions and overriding Example

```
public class Sub extends Base {  
    ...  
    public void g() {}  
    public void g() throws E1 {}  
    public void g() throws E2 {}  
    public void g() throws E3 {}  
    public void g() throws E1,E2 {}  
    public void g() throws E1,E3 {}  
    public void g() throws E2,E3 {}  
    public void g() throws E1,E2,E3 {}  
    ...  
}
```

Any of these is a
correct overring
of g

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 50

Exceptions and overriding Example

```
public class Sub extends Base {  
    ...  
    public void h() {}  
    public void h() throws E2 {}  
  
    public void h() throws E1 {}  
    public void h() throws E3 {}  
    public void h() throws E1,E2 {}  
    public void h() throws E1,E3 {}  
    public void h() throws E2,E3 {}  
    public void h() throws E1,E2,E3 {}  
}
```

Any of these is a
correct overring
of h

... but NONE
of these

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 51

Review

- Runtime errors arise for many reasons.
 - An inappropriate client call to a server object.
 - A server unable to fulfill a request.
 - Programming error in client and/or server.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 52

Review

- Runtime errors often lead to program failure.
- Defensive programming anticipates errors
 - in both client and server.
- Exceptions provide a reporting and recovery mechanism.

Object oriented programming, DAT042, DA2, 12/13, lp 1

Förel. 14 53