Creating Custom Exceptions

Esteban Herrera
JAVA ARCHITECT

@eh3rrera http://eherrera.net

Overview

Creating custom exceptions

When to create custom exceptions

Choosing between checked and unchecked exceptions

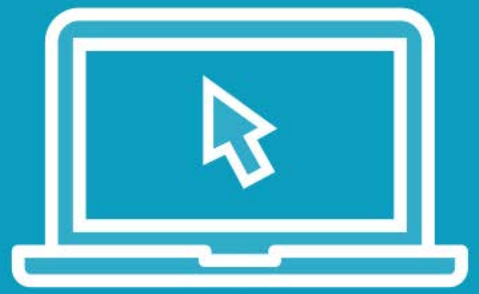
Package structure

Creating Custom Exceptions

Exceptions are objects

Like everything in Java

Demo



Creating a custom exception

Best Practices When Creating Exceptions

- Append the string Exception to the name
- Provide useful constructors and methods
- Inherit from either Exception or RuntimeException

When to Create Custom Exceptions

Do NOT create custom exceptions if they do NOT provide useful information

```
public class InvalidDateException extends Exception {

    public InvalidDateException() {
        super();
    }

    public InvalidDateException(String message) {
        super(message);
    }
}
```

```
throw new Exception("Invalid date");
```

```
throw new RuntimeException("Invalid date");
```

compact1, compact2, compact3

java.lang

Class Exception

java.lang.Object
    java.lang.Throwable
        java.lang.Exception

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CertificateException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalClassFormatException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, IOException, JAXBException, JMException, KeySelectorException, LambdaConversionException, LastOwnerException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ScriptException, ServerNotActiveException, SOAPException, SQLException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URIReferenceException, URISyntaxException, UserException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException

---

public class **Exception**
extends Throwable

The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

compact1, compact2, compact3

java.lang

Class RuntimeException

java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, CompletionException, ConcurrentModificationException, DataBindingException, DateTimeException, DOMException, EmptyStackException, EnumConstantNotPresentException, EventException, FileSystemAlreadyExistsException, FileSystemNotFoundException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, IllformedLocaleException, ImagingOpException, IncompleteAnnotationException, IndexOutOfBoundsException, JMRuntimeException, LSException, MalformedParameterizedTypeException, MalformedParametersException, MirroredTypesException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NoSuchMechanismException, NullPointerException, ProfileDataException, ProviderException, ProviderNotFoundException, RasterFormatException, RejectedExecutionException, SecurityException, SystemException, TypeConstraintException, TypeNotPresentException, UncheckedIOException, UndeclaredThrowableException, UnknownEntityException, UnmodifiableSetException, UnsupportedOperationException, WebServiceException, WrongMethodTypeException

---

```
public class RuntimeException
extends Exception
```

RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

RuntimeException and its subclasses are *unchecked exceptions*. Unchecked exceptions do *not* need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

ORACLE Java Documentation

The Java™ Tutorials

Creating Exception Classes

When faced with choosing the type of exception to throw, you can either use one written by someone else — the Java platform provides a lot of exception classes you can use — or you can write one of your own. You should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else's.

- Do you need an exception type that isn't represented by those in the Java platform?
- Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
- Does your code throw more than one related exception?
- If you use someone else's exceptions, will users have access to those exceptions? A similar question is, should your package be independent and self-contained?

An Ex

Suppos

he argument is less than 0 or more

http://bit.ly/creatingExceptions

When to Create a New Exception

Does an exception exist for the error?

Does the exception need special handling?

Do you need specific behavior or information?

Choosing Between Checked and Unchecked Exceptions

Not everything has to be black and white

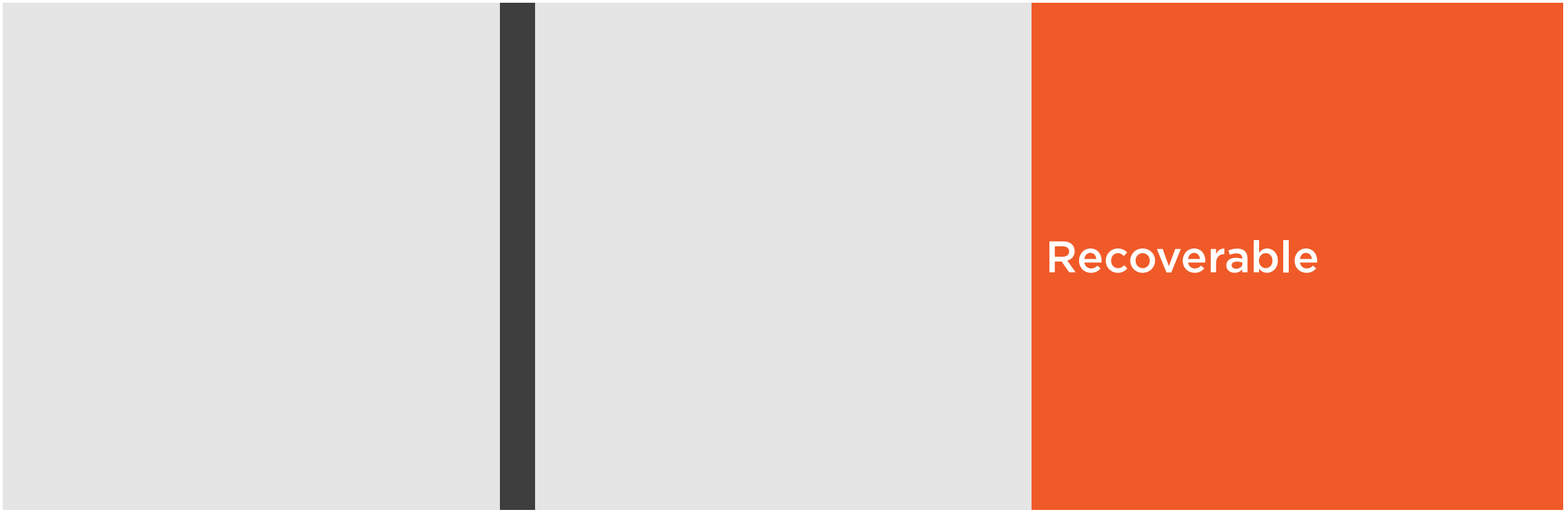
Exception Best Practices

Effective Java, 2nd edition

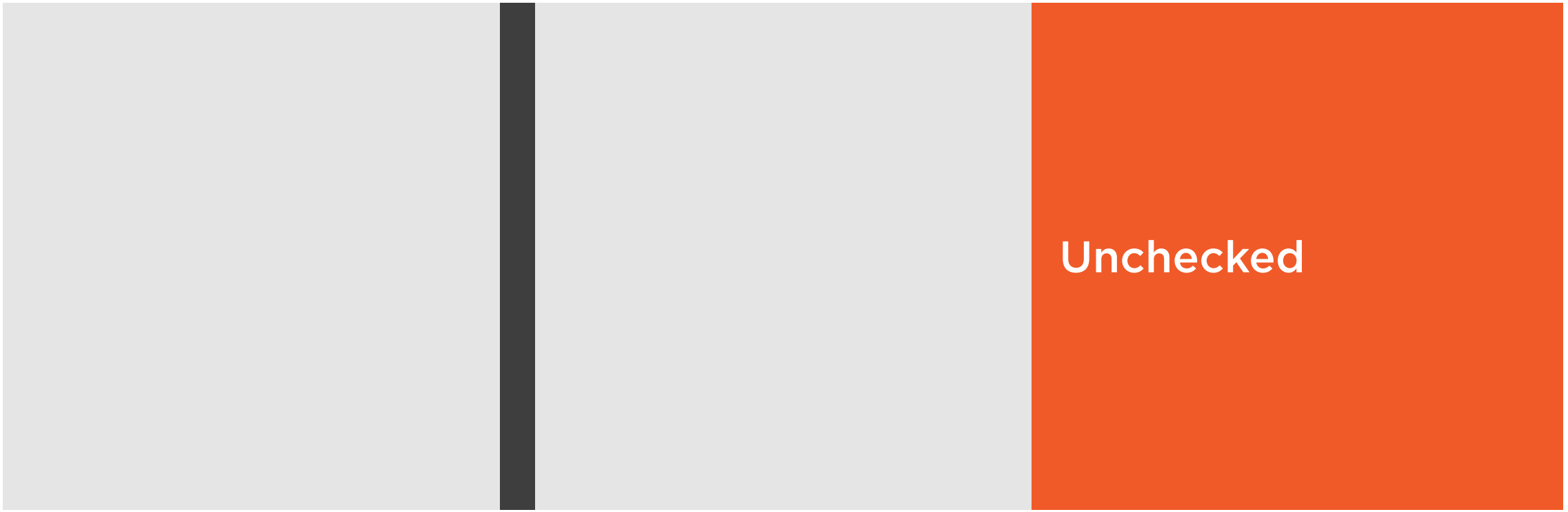
Use checked exceptions for recoverable conditions and runtime exceptions for programming errors.

Joshua Bloch, Effective Java, 2nd edition, Item 58.

Expected / Recoverable



Checked or Unchecked Exceptions



Avoid unnecessary use of checked exceptions.

Joshua Bloch, Effective Java, 2nd edition, Item 59.

```
try {
    // ...
} catch(CheckedException e) {
    e.printStackTrace();
}
```

Java Exception Mapping

Contingencies (checked exceptions)

Faults (unchecked exceptions)

http://bit.ly/effectiveJavaExceptions

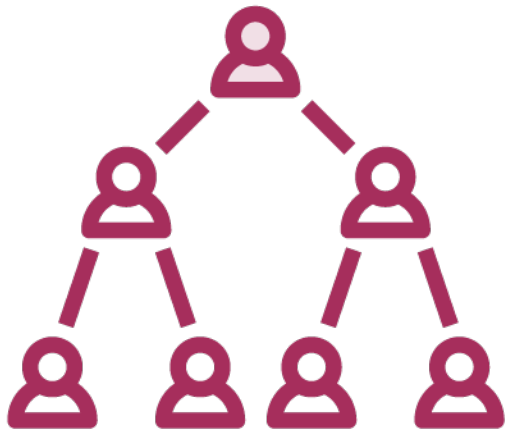
Guidelines

Checked Exceptions	Unchecked Exceptions
An expected error	A surprising error
It can happen sometimes	It should never happen
Business errors	Programming/hardware errors

Package Structure

Package Structure

Layer



com.company.controller

com.company.model

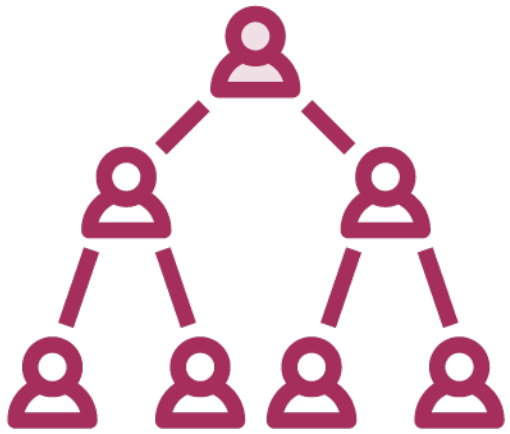
com.company.exception

- InvalidProductException
- CreditSupplierException
- UserNotFoundException

Package Structure

Layer

Feature



com.company.product

com.company.supplier

com.company.user

- UserController
- UserModel
- UserNotFoundException

Layer vs Feature

Layer	Feature
ProductService	UserController
SupplierService	UserModel
UserService	UserNotFoundException

Package by Feature Advantages

- Higher modularity
- Easier navigation and organization
- Minimizes scope

Summary

- Creating custom exceptions
- When to create custom exceptions
- Choosing between checked and unchecked exceptions
- Package structure