15-214

Principles of Software Construction: Objects, Design, and Concurrency

Principles of API Design

Fall 2014

Charlie Garrod Jonathan Aldrich

School of Computer Science
institute for SOFTWARE RESEARCH

Closely based on *How To Design A Good API and Why It Matters* by Josh Bloch

Administrivia

- Homework 4c due next Tuesday!
- Midterm exam next Thursday
  - Review session Tuesday, 5-7 p.m. in PH 100

Key concepts from Tuesday

Today: API design

- Introduction to APIs: Application Programming Interfaces
- An API design process
- Key design principle: Information hiding
- Concrete advice for user-centered design

Today: API design

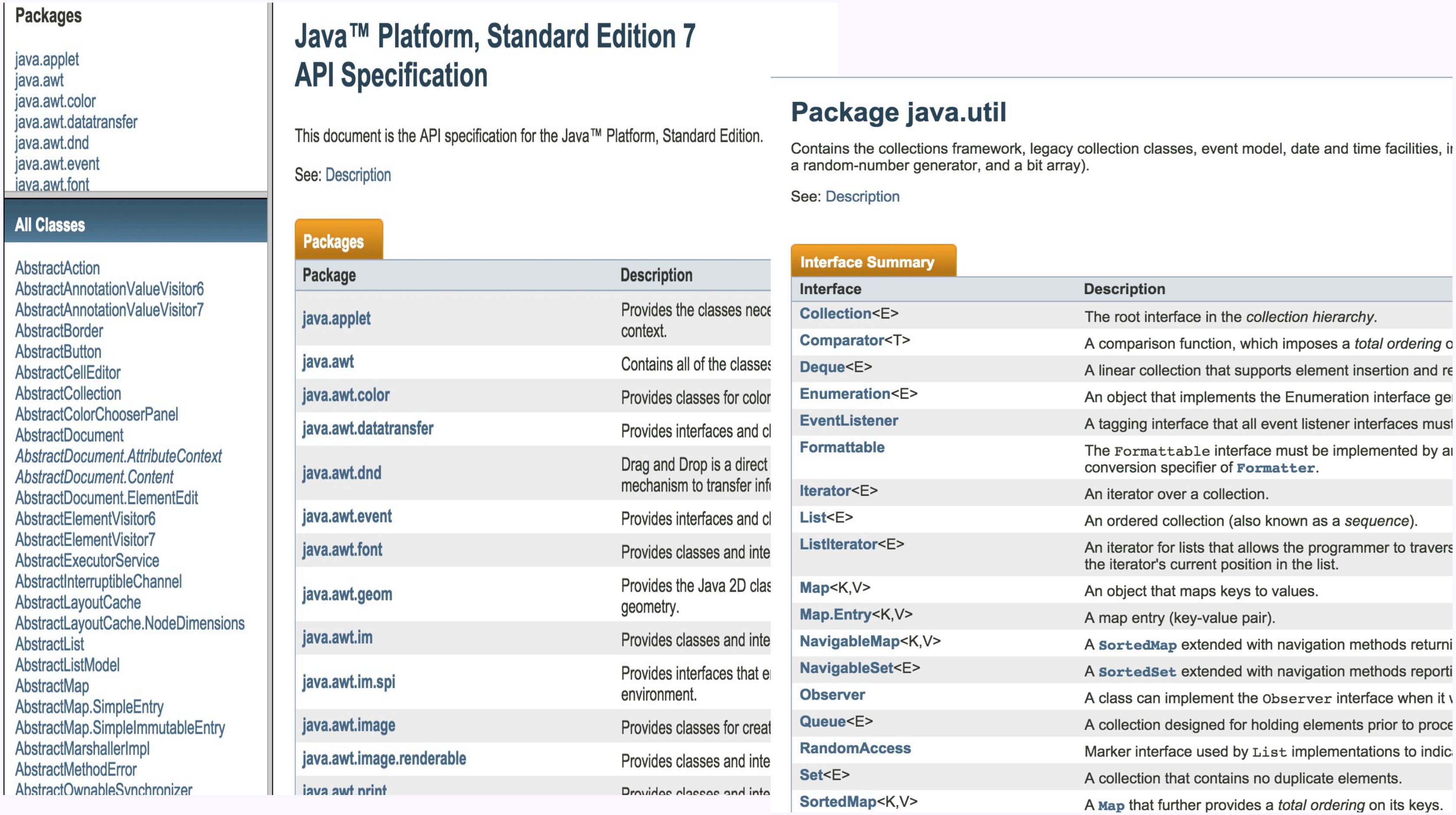
- Introduction to APIs: Application Programming Interfaces
- An API design process
- Key design principle: Information hiding
- Concrete advice for user-centered design
- Based heavily on "How to Design a Good API and Why it Matters by Josh Bloch"
  - If you have "Java" in your resume you should own *Effective Java*, our optional course textbook.

Learning goals for today

- Understand and be able to discuss the similarities and differences between API design and regular software design
  - Relationship between libraries, frameworks and API design
  - Information hiding as a key design principle
- Acknowledge, and plan for failures as a fundamental limitation on a design process
- Given a problem domain with use cases, be able to plan a coherent design process for an API for those use cases
  - "Rule of Threes"

API: Application Programming Interface

- An API defines the boundary between components/modules in a programmatic system



Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer

Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

Package	Description
java.applet	Provides the classes nece context.
java.awt	Contains all of the classes
java.awt.color	Provides classes for color
java.awt.datatransfer	Provides interfaces and cl
java.awt.dnd	Drag and Drop is a direct mechanism to transfer inf
java.awt.event	Provides interfaces and cl
java.awt.font	Provides classes and inte
java.awt.geom	Provides the Java 2D clas geometry.
java.awt.im	Provides classes and inte
java.awt.im.spi	Provides interfaces that e environment.
java.awt.image	Provides classes for creat
java.awt.image.renderable	Provides classes and inte
java.awt.print	Provides classes and inte

Package java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, i a random-number generator, and a bit array).

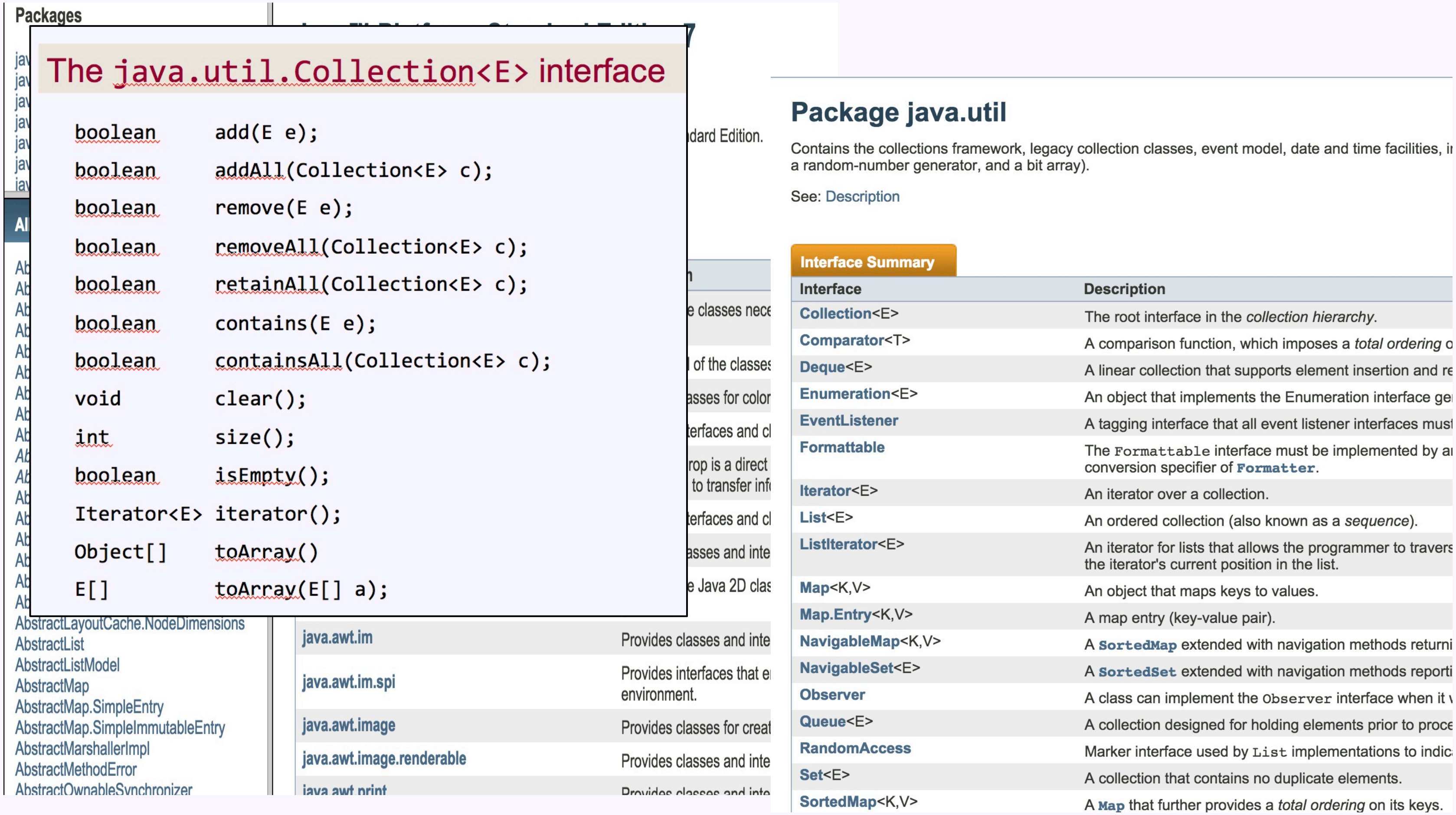
See: Description

Interface Summary

Interface	Description
Collection<E>	The root interface in the *collection hierarchy*.
Comparator<T>	A comparison function, which imposes a *total ordering* o
Deque<E>	A linear collection that supports element insertion and re
Enumeration<E>	An object that implements the Enumeration interface ge
EventListener	A tagging interface that all event listener interfaces must
Formattable	The `Formattable` interface must be implemented by a conversion specifier of `Formatter`.
Iterator<E>	An iterator over a collection.
List<E>	An ordered collection (also known as a *sequence*).
ListIterator<E>	An iterator for lists that allows the programmer to travers the iterator's current position in the list.
Map<K,V>	An object that maps keys to values.
Map.Entry<K,V>	A map entry (key-value pair).
NavigableMap<K,V>	A `SortedMap` extended with navigation methods returni
NavigableSet<E>	A `SortedSet` extended with navigation methods reporti
Observer	A class can implement the `Observer` interface when it
Queue<E>	A collection designed for holding elements prior to proce
RandomAccess	Marker interface used by `List` implementations to indic
Set<E>	A collection that contains no duplicate elements.
SortedMap<K,V>	A `Map` that further provides a *total ordering* on its keys.

API: Application Programming Interface

- An API defines the boundary between components/modules in a programmatic system



Packages

AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer

java.awt.im	Provides classes and inte
java.awt.im.spi	Provides interfaces that e environment.
java.awt.image	Provides classes for creat
java.awt.image.renderable	Provides classes and inte
java.awt.print	Provides classes and inte

The java.util.Collection<E> interface

```
boolean       add(E e);
boolean       addAll(Collection<E> c);
boolean       remove(E e);
boolean       removeAll(Collection<E> c);
boolean       retainAll(Collection<E> c);
boolean       contains(E e);
boolean       containsAll(Collection<E> c);
void          clear();
int           size();
boolean       isEmpty();
Iterator<E>   iterator();
Object[]      toArray()
E[]           toArray(E[] a);
```

Package java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, i a random-number generator, and a bit array).

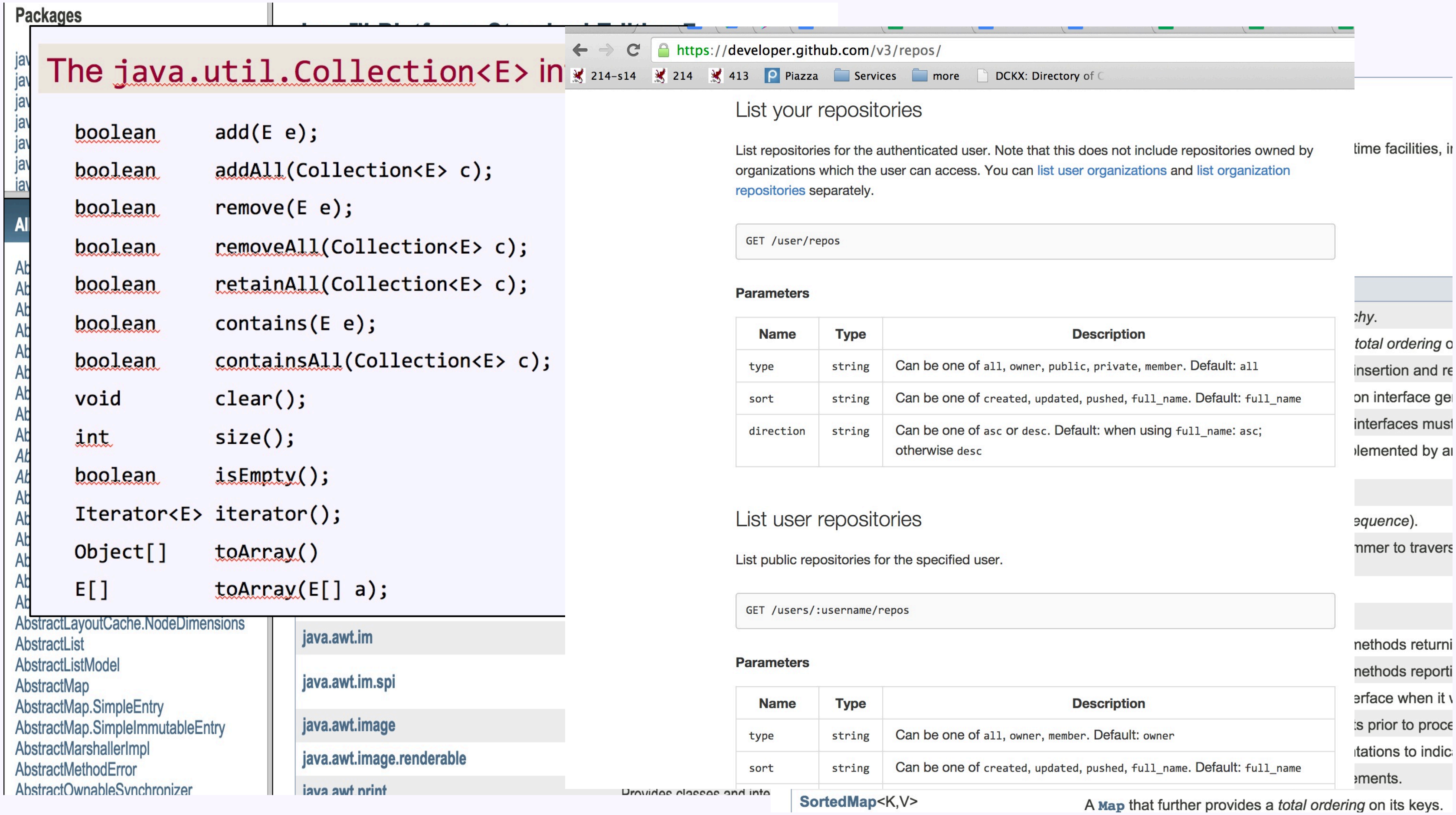
See: Description

Interface Summary

Interface	Description
Collection<E>	The root interface in the *collection hierarchy*.
Comparator<T>	A comparison function, which imposes a *total ordering* o
Deque<E>	A linear collection that supports element insertion and re
Enumeration<E>	An object that implements the Enumeration interface ge
EventListener	A tagging interface that all event listener interfaces must
Formattable	The `Formattable` interface must be implemented by a conversion specifier of `Formatter`.
Iterator<E>	An iterator over a collection.
List<E>	An ordered collection (also known as a *sequence*).
ListIterator<E>	An iterator for lists that allows the programmer to travers the iterator's current position in the list.
Map<K,V>	An object that maps keys to values.
Map.Entry<K,V>	A map entry (key-value pair).
NavigableMap<K,V>	A `SortedMap` extended with navigation methods return
NavigableSet<E>	A `SortedSet` extended with navigation methods reporti
Observer	A class can implement the `Observer` interface when it
Queue<E>	A collection designed for holding elements prior to proce
RandomAccess	Marker interface used by `List` implementations to indic
Set<E>	A collection that contains no duplicate elements.
SortedMap<K,V>	A `Map` that further provides a *total ordering* on its keys.

API: Application Programming Interface

- An API defines the boundary between components/modules in a programmatic system



Packages

AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer

java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderable

The java.util.Collection<E> in

```
boolean       add(E e);
boolean       addAll(Collection<E> c);
boolean       remove(E e);
boolean       removeAll(Collection<E> c);
boolean       retainAll(Collection<E> c);
boolean       contains(E e);
boolean       containsAll(Collection<E> c);
void          clear();
int           size();
boolean       isEmpty();
Iterator<E>   iterator();
Object[]      toArray()
E[]           toArray(E[] a);
```

https://developer.github.com/v3/repos/

List your repositories

List repositories for the authenticated user. Note that this does not include repositories owned by organizations which the user can access. You can list user organizations and list organization repositories separately.

```
GET /user/repos
```

Parameters

Name	Type	Description
type	string	Can be one of `all`, `owner`, `public`, `private`, `member`. Default: `all`
sort	string	Can be one of `created`, `updated`, `pushed`, `full_name`. Default: `full_name`
direction	string	Can be one of `asc` or `desc`. Default: when using `full_name`: `asc`; otherwise `desc`

List user repositories

List public repositories for the specified user.

```
GET /users/:username/repos
```

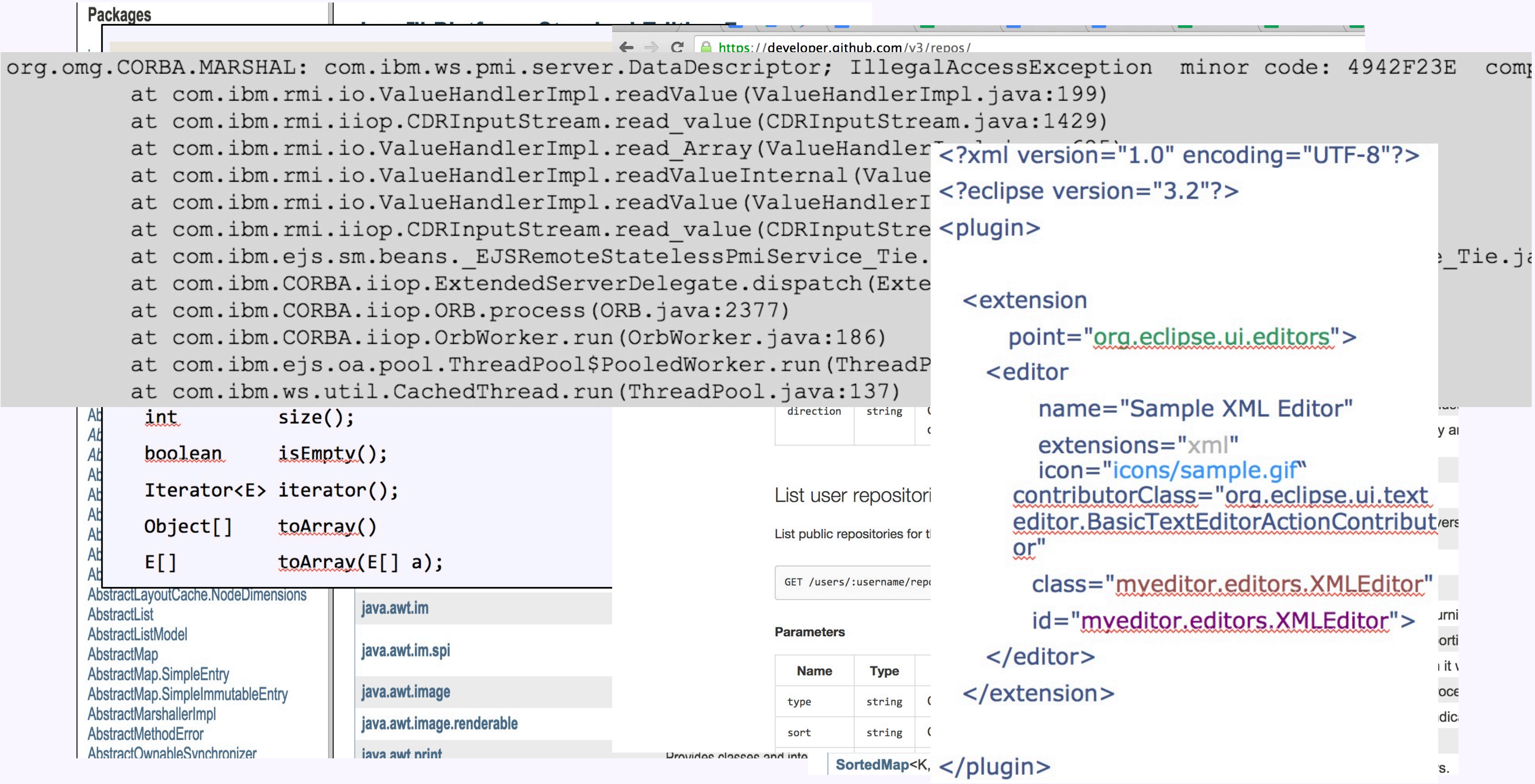
Parameters

Name	Type	Description
type	string	Can be one of `all`, `owner`, `member`. Default: `owner`
sort	string	Can be one of `created`, `updated`, `pushed`, `full_name`. Default: `full_name`

SortedMap<K,V> A Map that further provides a *total ordering* on its keys.

API: Application Programming Interface

- An API defines the boundary between components/modules in a programmatic system



Packages

https://developer.github.com/v3/repos/

```
org.omg.CORBA.MARSHAL: com.ibm.ws.pmi.server.DataDescriptor; IllegalAccessException  minor code: 4942F23E  comp
        at com.ibm.rmi.io.ValueHandlerImpl.readValue(ValueHandlerImpl.java:199)
        at com.ibm.rmi.iiop.CDRInputStream.read_value(CDRInputStream.java:1429)
        at com.ibm.rmi.io.ValueHandlerImpl.read_Array(ValueHandler
        at com.ibm.rmi.io.ValueHandlerImpl.readValueInternal(Value
        at com.ibm.rmi.io.ValueHandlerImpl.readValue(ValueHandlerI
        at com.ibm.rmi.iiop.CDRInputStream.read_value(CDRInputStre
        at com.ibm.ejs.sm.beans._EJSRemoteStatelessPmiService_Tie.
        at com.ibm.CORBA.iiop.ExtendedServerDelegate.dispatch(Exte
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:2377)
        at com.ibm.CORBA.iiop.OrbWorker.run(OrbWorker.java:186)
        at com.ibm.ejs.oa.pool.ThreadPool$PooledWorker.run(ThreadP
        at com.ibm.ws.util.CachedThread.run(ThreadPool.java:137)
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

  <extension
       point="org.eclipse.ui.editors">
    <editor
          name="Sample XML Editor"
          extensions="xml"
          icon="icons/sample.gif"
       contributorClass="org.eclipse.ui.text
       editor.BasicTextEditorActionContribut
       or"
          class="myeditor.editors.XMLEditor"
          id="myeditor.editors.XMLEditor">
    </editor>
  </extension>

</plugin>
```

```java
int          size();
boolean      isEmpty();
Iterator<E>  iterator();
Object[]     toArray()
E[]          toArray(E[] a);
```

List user repositori

List public repositories for t

```
GET /users/:username/repo
```

Parameters

Name	Type
direction	string
type	string
sort	string

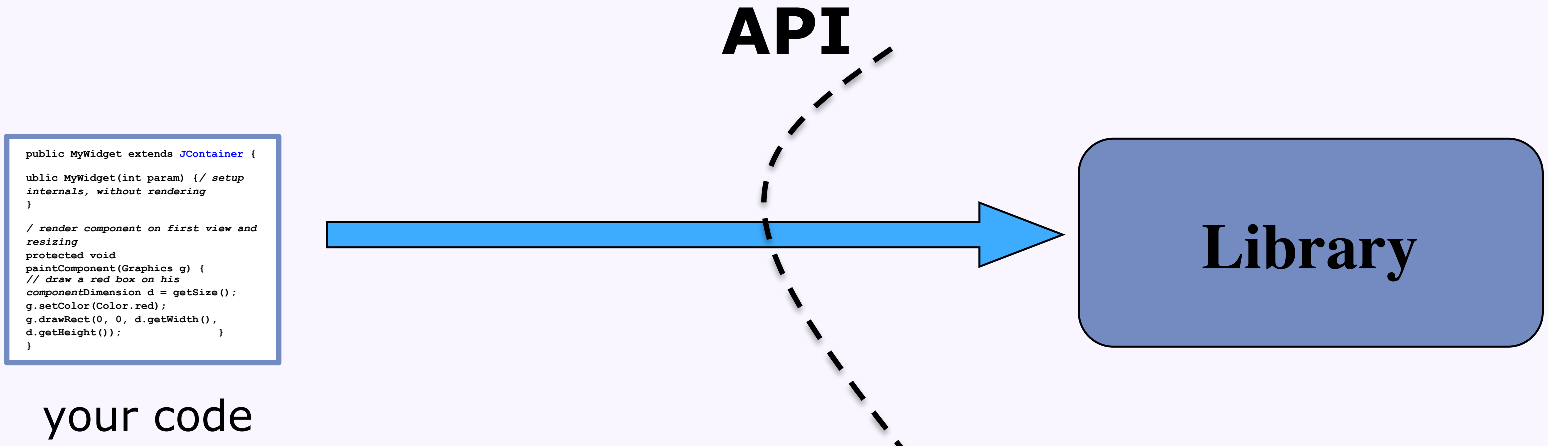
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer

java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderable
java.awt.print

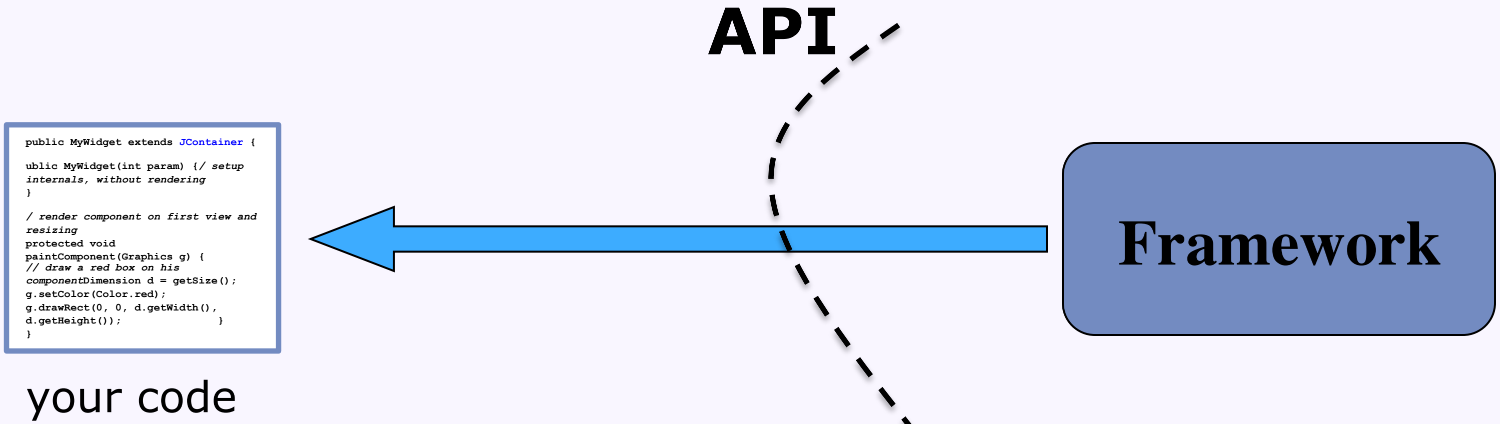
Provides classes and inte

SortedMap<K,

Libraries and frameworks both define APIs



```
public MyWidget extends JContainer {

ublic MyWidget(int param) {/ setup
internals, without rendering
}

/ render component on first view and
resizing
protected void
paintComponent(Graphics g) {
// draw a red box on his
componentDimension d = getSize();
g.setColor(Color.red);
g.drawRect(0, 0, d.getWidth(),
d.getHeight());                  }
}
```



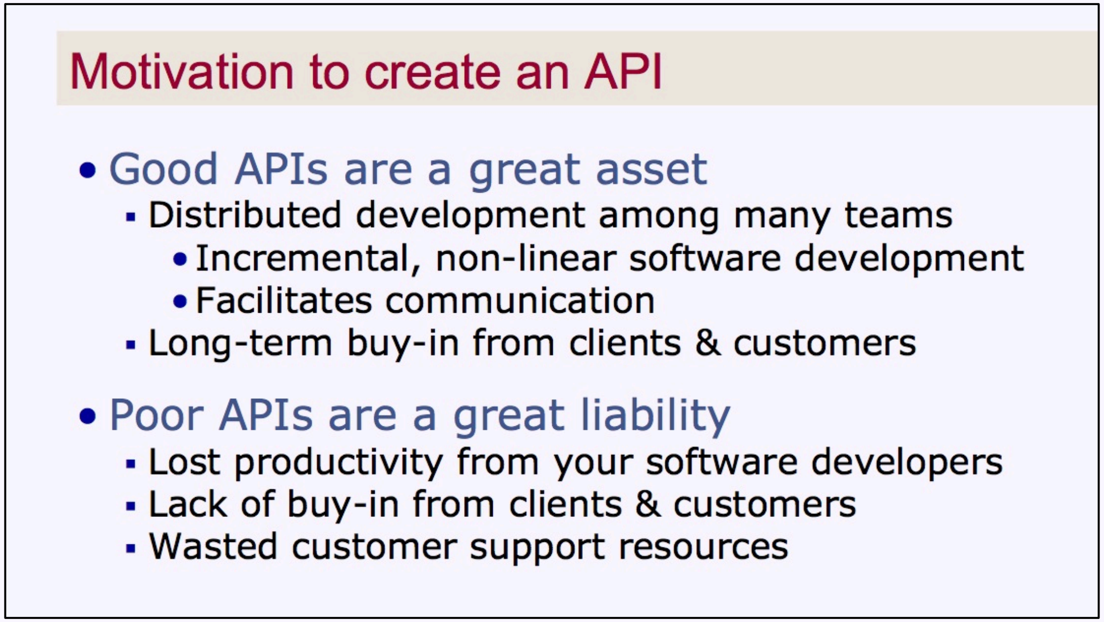
```
public MyWidget extends JContainer {

ublic MyWidget(int param) {/ setup
internals, without rendering
}

/ render component on first view and
resizing
protected void
paintComponent(Graphics g) {
// draw a red box on his
componentDimension d = getSize();
g.setColor(Color.red);
g.drawRect(0, 0, d.getWidth(),
d.getHeight());                  }
}
```

Motivation to create a public API

- Good APIs are a great asset
  - Distributed development among many teams
    - Incremental, non-linear software development
    - Facilitates communication
  - Long-term buy-in from clients & customers
- Poor APIs are a great liability
  - Lost productivity from your software developers
  - Lack of buy-in from clients & customers
  - Wasted customer support resources

Evolutionary problems: Public APIs are forever

- "One chance to get it right"
- You can add features, but never remove or change the behavioral contract for an existing feature



Motivation to create an API

- Good APIs are a great asset
  - Distributed development among many teams
    - Incremental, non-linear software development
    - Facilitates communication
  - Long-term buy-in from clients & customers
- Poor APIs are a great liability
  - Lost productivity from your software developers
  - Lack of buy-in from clients & customers
  - Wasted customer support resources

An API design process

- Define the scope of the API
  - Collect use-case stories, define requirements
  - Be skeptical
    - Distinguish true requirements from so-called solutions
    - "When in doubt, leave it out."
- Draft a specification, gather feedback, revise, and repeat
  - Keep it simple, short
- Code early, code often
  - Write *client code* before you implement the API

Respect the rule of three

- Via Will Tracz (via Josh Bloch), *Confessions of a Used Program Salesman*:
  - "If you write one, it probably won't support another."
  - "If you write two, it will support more with difficulty."
  - "If you write three, it will work fine."

Documenting an API

- APIs should be self-documenting
  - Good names drive good design
- Document religiously anyway
  - All public classes
  - All public methods
  - All public fields
  - All method parameters
  - Explicitly write behavioral specifications
- Documentation is integral to the design and development process

Key design principle: Information hiding

- "When in doubt, leave it out."

Documenting an API

- APIs should be self-documenting
  - Good names drive good design
- Document religiously anyway
  - All public classes
  - All public methods
  - All public fields
  - All method parameters
  - Explicitly write behavioral specifications
- Documentation is integral to the design and development process
- Do not document implementation details

Key design principle: Information hiding (2)

- Minimize the accessibility of classes, fields, and methods
  - "You can add features, but never remove or change the behavioral contract for an existing feature"

Key design principle: Information hiding (3)

- Use accessor methods, not public fields
  - Consider:

```
    public class Point {
        public double x;
        public double y;
    }
vs.
    public class Point {
        private double x;
        private double y;
        public double getX() { /* … */ }
        public double getY() { /* … */ }
    }
```

Key design principle: Information hiding (4)

- Prefer interfaces over abstract classes
  - Interfaces provide greater flexibility, avoid needless implementation details
  - Consider our earlier example:

```
public interface Point {
   public double get();
   public double getY();
}

public class PolarPoint() implements Point {
   private double r;       // Distance from origin.
   private double theta;   // Angle.
   public double getX() { return r*Math.cos(theta); }
   public double getY() { return r*Math.sin(theta); }
}
```

Key design principle: Information hiding (5)

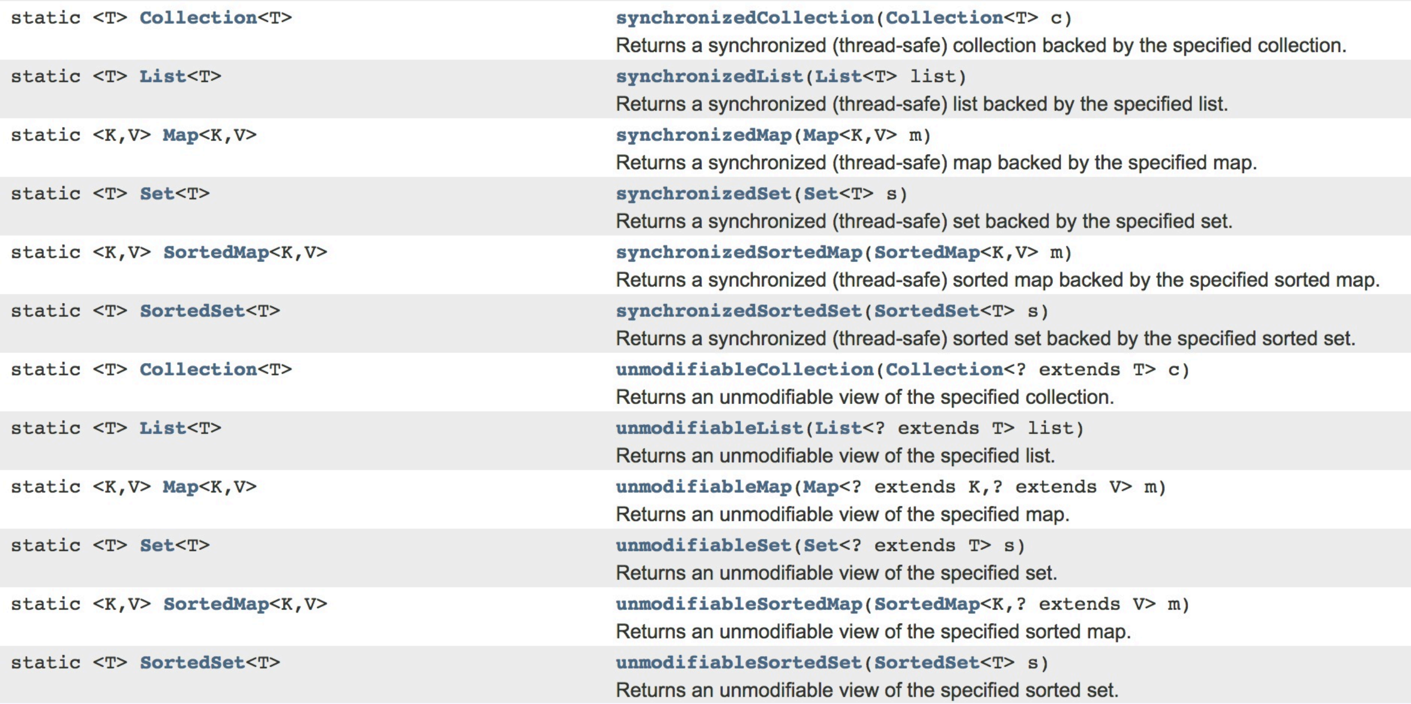
- Consider implementing a factory method instead of a constructor
    - Factory methods provide additional flexibility
        - Can be overridden
        - Can return instance of any subtype
            - Hides dynamic type of object
        - Can have a descriptive method name

Key design principle: Information hiding (6)

- Prevent subtle leaks of implementation details
  - Documentation
  - Implementation-specific return types
  - Implementation-specific exceptions
  - Output formats
  - `implements Serializable`

Minimize conceptual weight

- Conceptual weight: How many concepts must a programmer learn to use your API?
  - APIs should have a "high power-to-weight ratio"
- See `java.util.*`, `java.util.Collections`



`static <T> Collection<T>`	`synchronizedCollection(Collection<T> c)` Returns a synchronized (thread-safe) collection backed by the specified collection.
`static <T> List<T>`	`synchronizedList(List<T> list)` Returns a synchronized (thread-safe) list backed by the specified list.
`static <K,V> Map<K,V>`	`synchronizedMap(Map<K,V> m)` Returns a synchronized (thread-safe) map backed by the specified map.
`static <T> Set<T>`	`synchronizedSet(Set<T> s)` Returns a synchronized (thread-safe) set backed by the specified set.
`static <K,V> SortedMap<K,V>`	`synchronizedSortedMap(SortedMap<K,V> m)` Returns a synchronized (thread-safe) sorted map backed by the specified sorted map.
`static <T> SortedSet<T>`	`synchronizedSortedSet(SortedSet<T> s)` Returns a synchronized (thread-safe) sorted set backed by the specified sorted set.
`static <T> Collection<T>`	`unmodifiableCollection(Collection<? extends T> c)` Returns an unmodifiable view of the specified collection.
`static <T> List<T>`	`unmodifiableList(List<? extends T> list)` Returns an unmodifiable view of the specified list.
`static <K,V> Map<K,V>`	`unmodifiableMap(Map<? extends K,? extends V> m)` Returns an unmodifiable view of the specified map.
`static <T> Set<T>`	`unmodifiableSet(Set<? extends T> s)` Returns an unmodifiable view of the specified set.
`static <K,V> SortedMap<K,V>`	`unmodifiableSortedMap(SortedMap<K,? extends V> m)` Returns an unmodifiable view of the specified sorted map.
`static <T> SortedSet<T>`	`unmodifiableSortedSet(SortedSet<T> s)` Returns an unmodifiable view of the specified sorted set.

Apply principles of user-centered design

- Other programmers are your users
- e.g., "Principles of Universal Design"
    - Equitable use
    - Flexibility in use
    - Simple and intuitive use
    - Perceptible information
    - Tolerance for error
    - Low physical effort
    - Size and space for approach and use

Good names drive good design

- Do what you say you do:
  - "Don't violate the Principle of Least Astonishment"

```
public class Thread implements Runnable {
  // Tests whether current thread has been interrupted.
  // Clears the interrupted status of current thread.
  public static boolean interrupted();
}
```

Good names drive good design (2)

- Follow language- and platform-dependent conventions
  - Typographical:
    - `get_x()` vs. `getX()`
    - `timer` vs. `Timer`, `HTTPServlet` vs `HttpServlet`
    - `edu.cmu.cs.cs214`
  - Grammatical:
    - Nouns for classes
    - Nouns or adjectives for interfaces

Good names drive good design (3)

- Use clear, specific naming conventions
  - `getX()` and `setX()` for simple accessors and mutators
  - `isX()` for simple boolean accessors
  - `computeX()` for methods that perform computation
  - `createX()` or `newInstance()` for factory methods
  - `toX()` for methods that convert the type of an object
  - `asX()` for wrapper of the underlying object

Good names drive good design (4)

- Be consistent
  - `computeX()` vs. `generateX()`?
  - `deleteX()` vs. `removeX()`?

Do not violate Liskov's behavioral subtyping rules

- Use inheritance only for true subtypes
- Favor composition over inheritance

```
// A Properties instance maps Strings to Strings
public class Properties extends HashTable {
    public Object put(Object key, Object value);
    …
}

public class Properties {
    private final HashTable data = new HashTable();
    public String put(String key, String value) {
        data.put(key, value);
    }
    …
}
```

Minimize mutability

- Immutable objects are:
  - Inherently thread-safe
  - Freely shared without concern for side effects
  - Convenient building blocks for other objects
  - Can share internal implementation among instances
    - See `java.lang.String`

Minimize mutability

- Immutable objects are:
  - Inherently thread-safe
  - Freely shared without concern for side effects
  - Convenient building blocks for other objects
  - Can share internal implementation among instances
    - See `java.lang.String`
- Mutable objects require careful management of visibility and side effects
  - e.g. `Component.getSize()` returns a mutable `Dimension`
- Document mutability
  - Carefully describe state space

Overload method names judiciously

- Avoid ambiguous overloads for subtypes
  - Recall the subtleties of method dispatch:

```
public class Point() {
    private int x;
    private int y;
    public boolean equals(Point p) {
        return this.x == p.x && this.y == p.y;
    }
}
```

- If you must be ambiguous, implement consistent behavior

```
public class TreeSet implements SortedSet {
  public TreeSet(Collection c);  // Ignores order.
  public TreeSet(SortedSet s);   // Respects order.
}
```

Use consistent parameter ordering

- An egregious example from C:

```
char* strncpy(char* dest, char* src,  size_t n);
void    bcopy(void* src,  void* dest, size_t n);
```

Avoid long lists of parameters

- Especially avoid parameter lists with repeated parameters of the same type

```
HWND CreateWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName,
  DWORD dwStyle, int x, int y, int nWidth, int nHeight,
  HWND hWndParent, HMENU hMenu, HINSTANCE hInstance,
  LPVOID lpParam);
```

- Break up the method or use a helper class to hold parameters instead

Fail fast

- Report errors as soon as they are detectable
  - Check preconditions at the beginning of each method
  - Avoid dynamic type casts, run-time type-checking

```
// A Properties instance maps Strings to Strings
public class Properties extends HashTable {
  public Object put(Object key, Object value);

  // Throws ClassCastException if this instance
  // contains any keys or values that are not Strings
  public void save(OutputStream out, String comments);
}
```

Avoid behavior that demands special processing

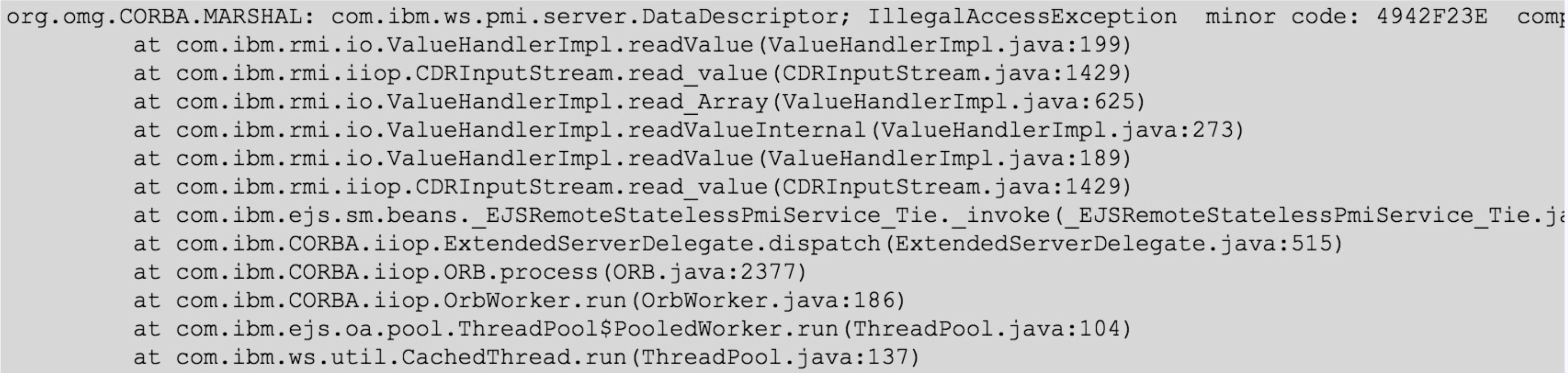
- Do not return `null` to indicate an empty value
  - e.g., Use an empty `Collection` or array instead
- Do not return `null` to indicate an error
  - Use an exception instead
- Do not return a `String` if a better type exists
- Do not use exceptions for normal behavior
- Avoid checked exceptions if possible

```
try {
   Foo f = (Foo) g.clone();
} catch (CloneNotSupportedException e) {
   // Do nothing. This exception can't happen.
}
```

Don't let your output become your de facto API

- Document the fact that output formats may evolve in the future
- Provide programmatic access to all data available in string form

```
public class Throwable {
  public void printStackTrace(PrintStream s);
}
```



```
org.omg.CORBA.MARSHAL: com.ibm.ws.pmi.server.DataDescriptor; IllegalAccessException  minor code: 4942F23E  comp
         at com.ibm.rmi.io.ValueHandlerImpl.readValue(ValueHandlerImpl.java:199)
         at com.ibm.rmi.iiop.CDRInputStream.read_value(CDRInputStream.java:1429)
         at com.ibm.rmi.io.ValueHandlerImpl.read_Array(ValueHandlerImpl.java:625)
         at com.ibm.rmi.io.ValueHandlerImpl.readValueInternal(ValueHandlerImpl.java:273)
         at com.ibm.rmi.io.ValueHandlerImpl.readValue(ValueHandlerImpl.java:189)
         at com.ibm.rmi.iiop.CDRInputStream.read_value(CDRInputStream.java:1429)
         at com.ibm.ejs.sm.beans._EJSRemoteStatelessPmiService_Tie._invoke(_EJSRemoteStatelessPmiService_Tie.ja
         at com.ibm.CORBA.iiop.ExtendedServerDelegate.dispatch(ExtendedServerDelegate.java:515)
         at com.ibm.CORBA.iiop.ORB.process(ORB.java:2377)
         at com.ibm.CORBA.iiop.OrbWorker.run(OrbWorker.java:186)
         at com.ibm.ejs.oa.pool.ThreadPool$PooledWorker.run(ThreadPool.java:104)
         at com.ibm.ws.util.CachedThread.run(ThreadPool.java:137)
```

Don't let your output become your de facto API

- Document the fact that output formats may evolve in the future
- Provide programmatic access to all data available in string form

```
public class Throwable {
  public void printStackTrace(PrintStream s);
  public StackTraceElement[] getStackTrace();
}

public final class StackTraceElement {
  public String  getFileName();
  public int     getLineNumber();
  public String  getClassName();
  public String  getMethodName();
  public boolean isNativeMethod();
}
```

Summary

- Accept the fact that you, and others, will make mistakes
  - Use your API as you design it
  - Get feedback from others
  - Hide information to give yourself maximum flexibility later
  - Design for inattentive, hurried users
  - Document religiously