Getting to know Java

AGENDA

Agenda

- How does Java Evolve?
- Java SE 6 standard packages
- Open Source Libraries for Java

HOW DOES JAVA EVOLVE?

The Java Community Process

- Java evolves through the JCP
- JCP
  - Java Community Process
  - http://jcp.org/
  - Process for adding/updating new specs to the Java platform
- New/updated specs for the Java platform are proposed through JSRs

Java Specification Requests (1/3)

- JSR
  - Java Specification Request
  - Describes a proposed and, then, a finalized spec to be added to the Java platform
  - Describes the JCP to use to achieve its goal

Java Specification Requests (2/3)

- The result of a JSR is generally made up of
  - A finalized spec (API)
  - A RI (Reference Implementation) for this spec
  - A TCK (Technology Compatibility Kit) to ensure other implementations comply with the API

Java Specification Requests (3/3)

- Some recent JSRs
  - JSR 221: Java Database Connectivity (JDBC) 4.0
  - JSR 241: The Groovy Programming Language
  - JSR 293: Location API 2.0 for Java ME
  - JSR 336: Java SE 7 Release Contents
  - JSR 337: Java SE 8 Release Contents

JAVA SE 6 STANDARD PACKAGES

java.applet, java.awt

- `java.applet`
  - Allows creating applets, which are small applications embedded into a browser

- `java.awt`
  - Abstract Window Toolkit
  - Set of classes allowing to build basic (and native) GUIs

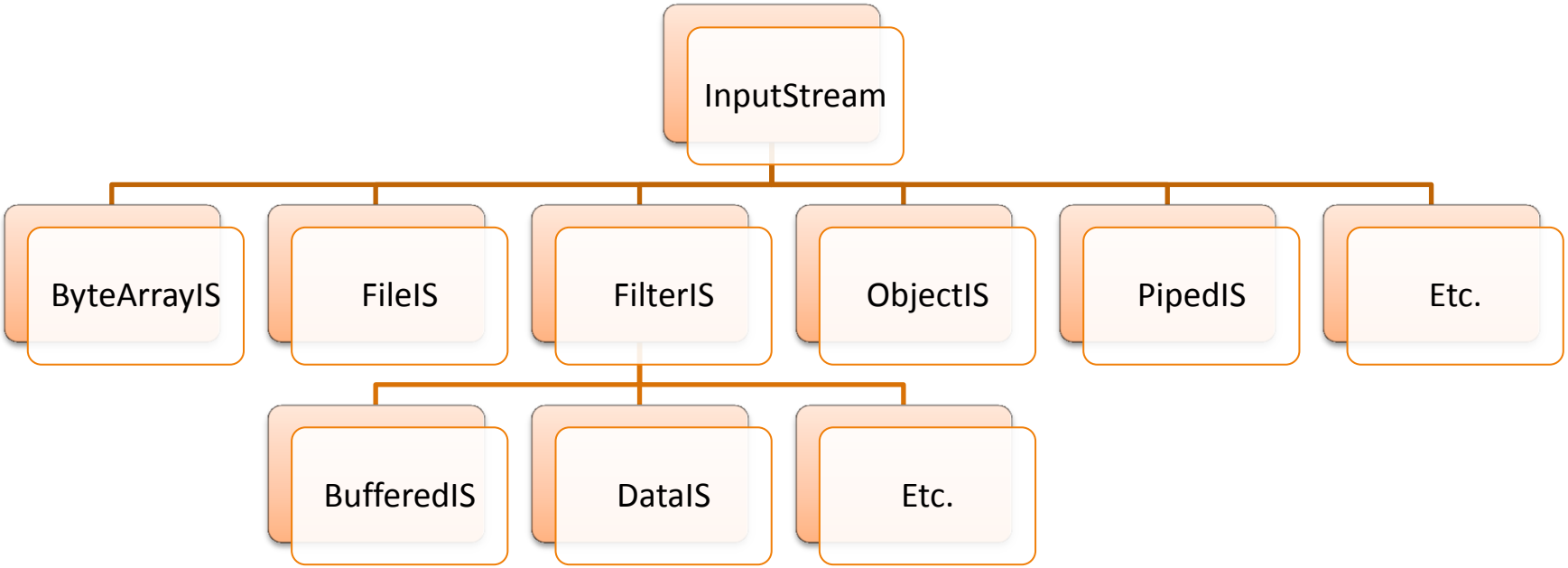
The JavaBeans API

- `java.beans`
  - Provides an API to work with JavaBeans
  - JavaBean
    - Serializable Java object, with a default constructor and getters and setters
    - Technology initially designed to visually build applications by assembling JavaBeans together
    - Reminder: POJOs (Plain Old Java Objects) are more that just beans

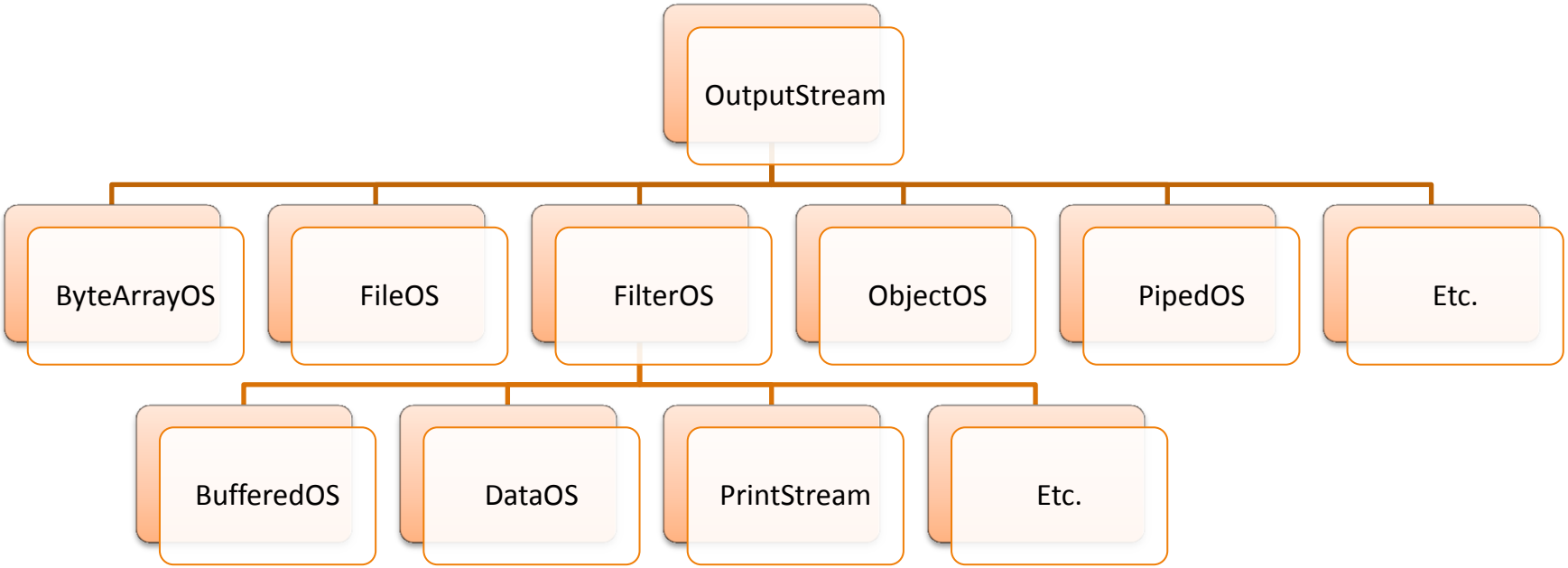
Handling inputs/outputs

- `java.io`
  - Provides all the necessary classes for handling I/O
  - Important classes:
    - `InputStream`: Represents a input stream of `bytes`
    - `OutputStream`: Represents an output stream of `bytes`
    - `Reader`: Allows reading character streams
    - `Writer`: Allows writing character streams

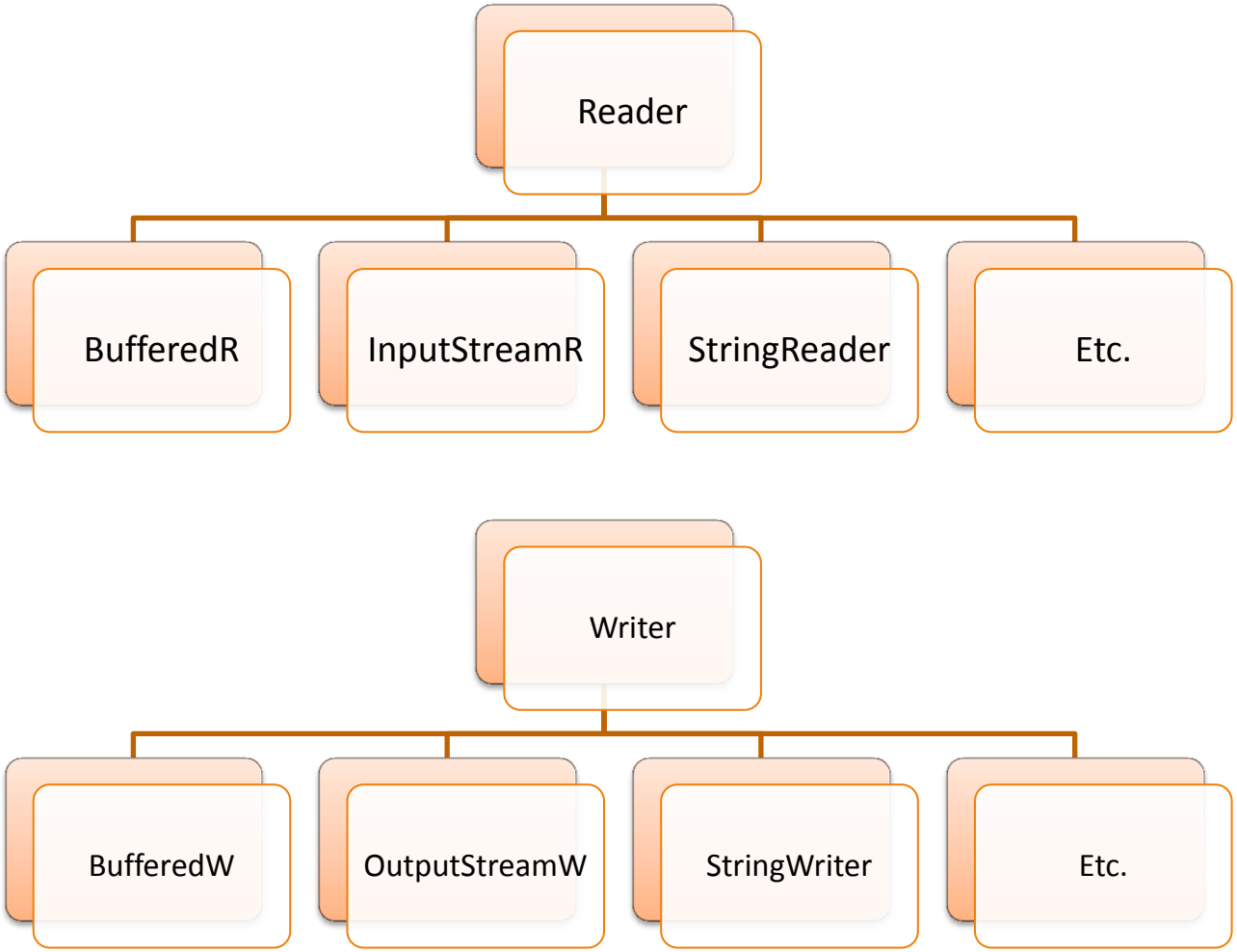
Input streams: Class hierarchy



Output streams: Class hierarchy



Readers/writers



Handling inputs/outputs

```
BufferedReader reader =
  new BufferedReader(
    new InputStreamReader(System.in));
String input = reader.readLine();

PipedInputStream i = new PipedInputStream();
PipedOutputStream o =
  new PipedOutputStream(i);
o.write(/* some data */); // one thread
data = i.read();          // another thread
```

Handling inputs/outputs

- `java.io` *(contd)*
  - Important classes:
    - `RandomAccessFile`: Allows improving file reading/writing
    - `Console`: Allows interacting with the console associated with the JVM (if any)

Handling inputs/outputs

```
Console console = System.console();
if(console != null) {
  Reader reader = console.reader();
  char c = reader.read();
  while(c != -1) {
    // …
    c = reader.read();
  }
}
```

java.lang and java.math

- `java.lang` (and `java.math`)
  - The most important Java package!
  - Contains classes that match with all Java's primitive types (`Integer` for `int`, `Double` for `double`, etc.)
  - `java.math.BigDecimal`: Very important class when precision matters
  - Features the `Thread` API
  - `ThreadLocal` allows playing with per-thread variables (vs. static or instance ones)

Working with primitives and strings

```
// string to int
int score = Integer.parseInt("20");

// int to string
String score = Integer.toString(20);
// NB: equivalent but better than
// String score = "" + i;
```

Threads

```
Thread thread = new Thread() {
  public void run() {
    // do something in a dedicated thread
  }
};
thread.start();
// do something else in the main thread
thread.join(); // wait for the other thread
               // to complete
```

Threads

```
private static int nextId = 0;
ThreadLocal threadId = new ThreadLocal() {
  protected
    synchronized Object initialValue() {
      return new Integer(nextId++);
  }
};
// each thread can access its unique ID
// using:
// ((Integer) (threadId.get())).intValue()
```

java.lang and java.math

- `java.lang` (and `java.math`) *(contd)*
  - Other important classes:
    - `Class`/`ClassLoader`
    - `Runtime`/`System`: Allows executing processes, invoking GC (which is a bad idea), getting environment properties, etc.
    - `StringBuffer` (synchronized) and `StringBuilder` (not synchronized) allow processing strings efficiently

Working with Class/ClassLoader

```
InputStream in =  getClass()
   .getResourceAsStream("myresource");

Object o = Class.forName("fr.isima.MyClass")
   .newInstance();

Object o = Thread.getCurrentThread()
   .getContextClassLoader()
   .loadClass("fr.isima.MyClass")
   .newInstance();
```

java.lang and java.math

```
Runtime.getRuntime().exec(
  "echo Hello, World!");

String s =
  s1 + " " + s2 + " " + s3;

StringBuilder sb =
  new StringBuilder();
sb.append(s1).append(' ').append(s2);
sb.append(' ').append(s3);
```

Annotations

- `java.lang.annotation` and `javax.annotation`
  - Allows building your own annotations and your own annotation processors
- `java.lang.instrument`
  - Helps building Java agents which are aimed at instrumenting Java applications

Working with references

- `java.lang.ref`
  - Provides classes that wrap objects, allowing to handle the memory
  - Example: `SoftReference`
    - A `SoftReference` is kept in memory as long as there is memory available in the JVM
    - Just before the JVM goes out of memory (`OutOfMemoryError`), `SoftReference`s are garbage-collected

Soft references

```
MyObject o = new MyObject(…);
SoftReference ref =
  new SoftReference(o);
// …
if(ref.get() != null) {
  o = ref.get();
}
else {
  o = new MyObject(…);
}
```

Working with reflection

- `java.lang.reflect`
  - Part of the reflection API (cf. previous lesson)
  - `Array`: Set of static methods for working with arrays
  - `Constructor`: Represents one constructor for a given class
  - `Field`: Represents one field (a static field or an attribute) for a given class

Working with reflection

- `java.lang.reflect` *(contd)*
  - `Method`: Represents one method for a given class
  - `Modifier`: Set of static methods (`isFinal`, `isPublic`, etc.) to work with modifiers
  - A modifier is represented as an `int`

The Reflection API

```
for(Method m: MyClass.class.getMethods()) {
   if(m.getName().startsWith("get")) {
      // this is a getter: display its value
      // assuming toString() is clean
      System.out.println(m.invoke(anObject));
   }
}
```

Networking, new I/O

- `java.net`/`javax.net`
  - Provides classes to build network-enabled applications
- `java.nio`
  - NIO stands for New I/O
  - Provides classes which improves I/O operations
  - Is used behind-the-scene by the `java.io` classes

RMI, security

- `java.rmi/javax.rmi`
  - RMI stands for Remote Method Invokation (that said, all is said)
- `java.security/javax.security/javax.crypto`
  - The first one is the security framework of Java
  - Allows working with `Permission`s, `KeyStore`s, `Policy`s, etc.
  - Provides classes for cryptographic operations

java.sql

- `java.sql/javax.sql`
  - The standard packages for performing SQL operations from within Java

java.sql

```
Class.forName(
  "oracle.jdbc.driver.OracleDriver");
Connection conn =
  DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:XE",
    "scott","tiger");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(
  "SELECT ENAME FROM EMP");
// continued on next slide
```

java.sql

```
while(rs.next()) {
   System.out.println(
     "Employee name: "
     + rs.getString("ENAME"));
}
conn.close();
```

java.text

- Set of classes to work with text, dates, etc.

```
DateFormat f =
   new SimpleDateFormat("dd/MM/yyyy");
Date d = f.parse("23/06/1999");
```

java.util

- `java.util`
  - The second most important Java package
  - Provides the developer with:
    - Timer API allowing to schedule `TimerTask`s to be performed in the JVM
    - Collection- and map-based APIs (`List`s, `Vector`s, `Map`s, `Iterator`s, `Enumeration`s, etc.)

Collections

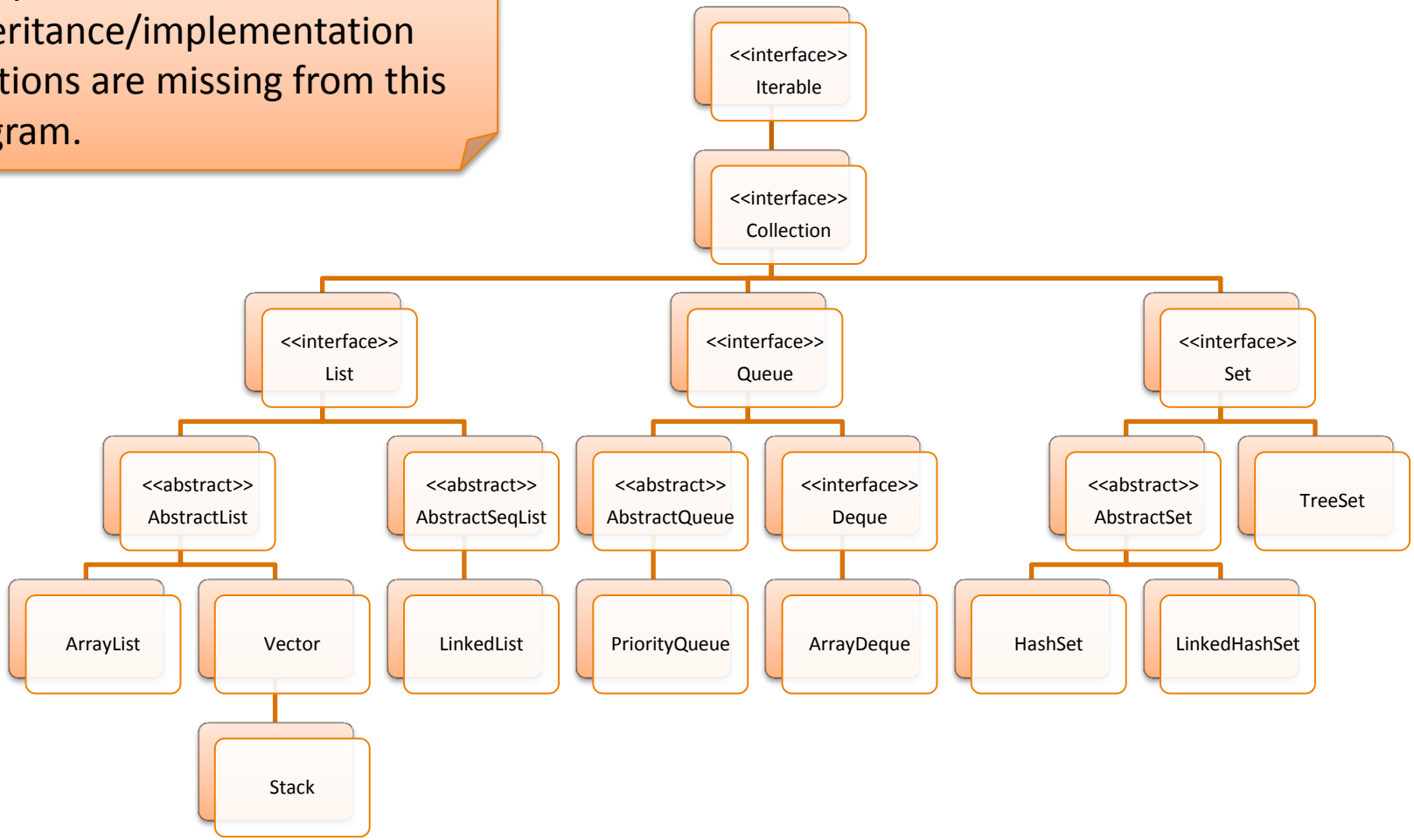
- `Iterable` interface: Defines the `iterator()` method to work with `Iterator`s
- `Collection`: Root interface for all collections
  - Methods: `add`, `contains`, `remove`, `toArray`, etc.
- Some important abstract collections:
  - `List`: Sequence of elements (possibly duplicated)
  - `Queue`
  - `Deque`: Double-ended queue
  - `Set`: Collection with no duplicates

Collections

- Concrete collections implementation:
  - `ArrayDeque`: Queue back by a resizable array
  - `ArrayList`: List backed by a resizable array
  - `HashSet`: Set backed by a hash map
  - `LinkedHashSet`
  - `LinkedList`
  - `PriorityQueue`: Queue based on a heap
  - `Stack`: LIFO synchronized stack (prefer `Deque`)
  - `TreeSet`: Set backed by a `TreeMap` (red-black tree)
  - `Vector`: Synchronized growable array

Collections: Class hierarchy

Many classes and inheritance/implementation relations are missing from this diagram.

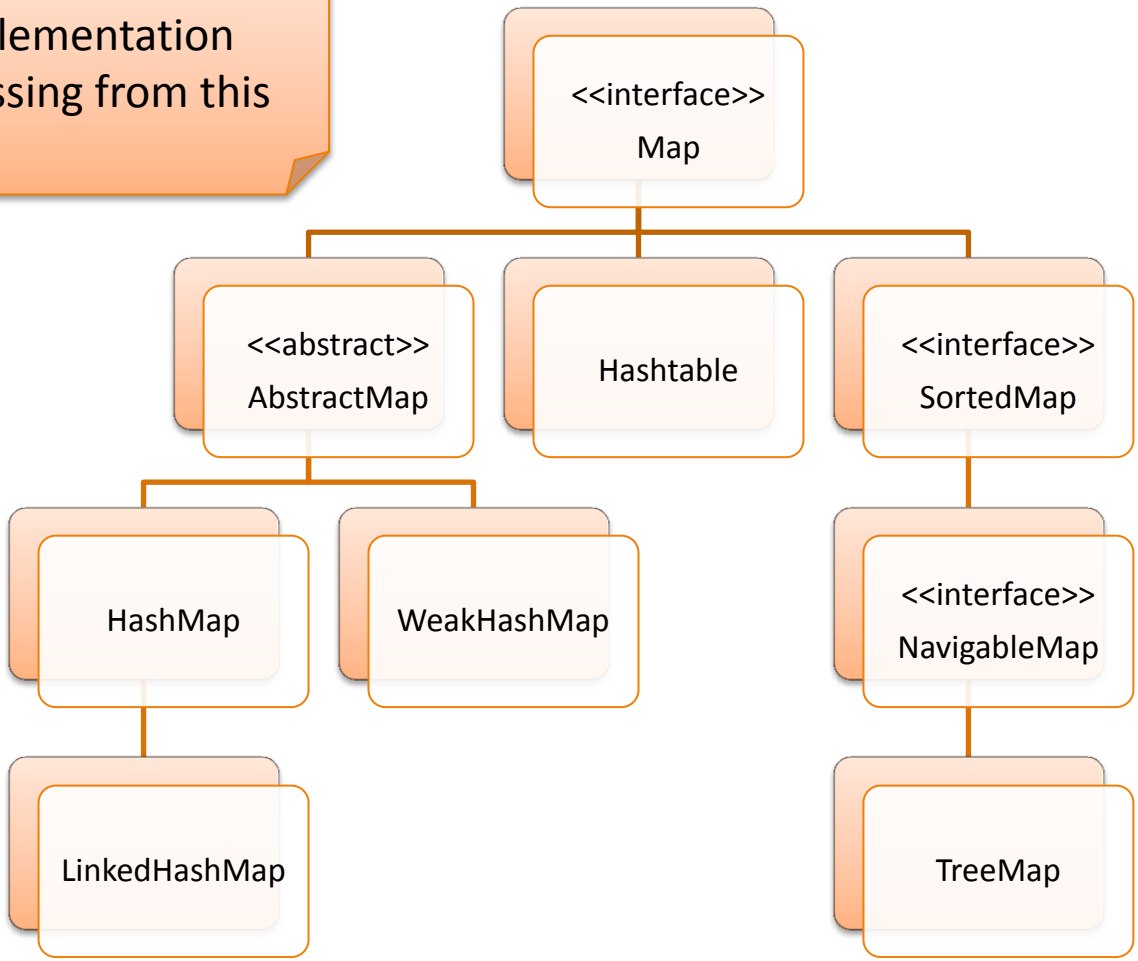


Maps

- `Map`: Root interface for maps
- `Map.Entry`: Represents an entry in a map
- Some concrete implementations:
  - `HashMap`: Default choice when working with maps
  - `Hashtable`: Synchronized map
  - `LinkedHashMap`: `HashMap` based on a linked list
  - `TreeMap`: Red-black tree (self-balancing binary search tree)

Maps: Class hierarchy

Many classes and inheritance/implementation relations are missing from this diagram.



Collections/Maps

- One important thing:
  - Your must work as much as possible with interfaces or abstract classes
  - Returning a concrete implementation is bad
  - `ArrayList list = new ArrayList(); // bad`
  - `List list = new ArrayList(); // good`

Collections/Maps

- Other important classes:
  - `Arrays` offers nice static methods, e.g.:
    - `asList()`
    - `copyOf()`
    - `equals()`
    - `sort()`: Uses quick-sort
  - `Collections` does the same, for example:
    - `reverse()`
    - `shuffle()`
    - `sort()`: Uses merge-sort

Collections/Maps

```
List<String> l =
  Arrays.asList("ab", "cd", "ef");
Iterator<String> it = l.iterator();
while(it.hasNext()) {
  String s = it.next();
}
```

Concurrency

- `java.util.concurrent`
  - Concurrent API to handle concurrency in applications
  - `Future`: Interface representing the result of an asynchronous computation
  - `FutureTask`: Implementation of `Future` and `Runnable`
  - `Callable`: ~`Runnable` which returns a result

Concurrency

- `java.util.concurrent` *(contd)*
  - `Executor`: Interface for executing `Runnable`s
  - `ExecutorService`: Specialized `Executor` interface for working with `Callable`s and `Future`s
  - `ScheduledExecutorService`: Specialized `ExecutorService` for working with delays/periods

Concurrency

- `java.util.concurrent` *(contd)*
  - `ThreadPoolExecutor`: Concrete `ExecutorService` working with a pool of threads
  - `ScheduledThreadPoolExecutor`: `ThreadPoolExecutor` with scheduling
  - `Executors`: Set of static methods which will save a lot of time and code

Concurrency

```
Future<String> future =
  Executors.newSingleThreadExecutor()
    .submit(new Callable<String>() {
      public String call() {
        // code which returns a String
      }
    });
String result =
  future.get(1, TimeUnit.MINUTES);
```

Concurrency and collections

- `java.util.concurrent` provides concurrency-enabled collections, e.g.:
  - `BlockingQueue` and `BlockingDeque` (interfaces)
  - `ArrayBlockingQueue`
  - `ConcurrentMap` (interface)
  - `ConcurrentHashMap`
  - `CopyOnWriteArrayList`

Concurrency and atomicity

- `java.util.concurrent.atomic` provides atomic classes (lock-free and thread safe):
  - `AtomicBoolean`
  - `AtomicInteger`
  - `AtomicIntegerArray`
  - `AtomicLong`
  - `AtomicLongArray`
  - Etc.
  - They don't replace primitives but must be used for flags, counters, sequences, etc.

java.util.logging, java.util.regex

- `java.util.logging`
  - Used to manage logging in applications (no more `System.out.println()` in your code!)
- `java.util.regex`
  - Provides two classes to work with regular expressions

java.util.logging, java.util.regex

```
static final Logger LOGGER =
  Logger.getLogger(
    MyObject.class.getName());
LOGGER.finest("blabla");

Pattern p = Pattern.compile(".*=.*");
Matcher m = p.matcher("name=value");
if(m.matches()) {
  // …
}
```

java.util.ServiceLoader

- `ServiceLoader`
  - Features static methods for loading a class implementing a service
  - A service is generally defined by an interface or an abstract class
  - The implementation of a service is defined in the following file: `META-INF/services/<interface name>`
  - Ex.: `META-INF/services/fr.isima.MyService` may contain the following line: `fr.isima.MyServiceImpl`

java.util.ServiceLoader

```
package fr.isima;
public interface MyService {
   void doThis();
}
public class MyServiceImpl
   implements MyService {
   public void doThis() {
      // some code
   }
}
```

java.util.ServiceLoader

```
// the following line triggers the reading
// of META-INF/services/fr.isima.MyService
ServiceLoader<MyService> loader
   = ServiceLoader.load(MyService.class);

// doThis is invoked for each impl listed
// in META-INF/services/fr.isima.MyService
for(MyService service : loader) {
   service.doThis();
}
```

javax.management, javax.print

- `javax.management`
  - This is the JMX (Java Management Extensions) API
  - Allows building applications to manage and monitor Java-based applications (e.g. JBoss AS)
- `javax.print`
  - This is the Java Print Service API

Scripting with Java

- `javax.script`
  - Because Java is more than just a programming language!
  - Used to build Java scripting engines for running scripts within the JVM
  - Some script languages for the JVM:
    - Groovy
    - Rhino (JavaScript)
    - Jython (Python)
    - JRuby (Ruby)

Scripting with Java

```
ScriptEngineManager factory =
  new ScriptEngineManager();
ScriptEngine engine =
  factory.getEngineByName("Groovy");
engine.eval("println('Hello, World')");
```

Swing

- `javax.swing`
  - Fully Java-based GUI components
  - Tip: To build GUIs, consider using Eclipse SWT (Standard Widget Toolkit)
    - Native-based, like AWT
    - Provides advanced components, like Swing
    - Most successful SWT-based application: Eclipse

Working with XML

- `javax.xml`/`org.w3c.dom`/`org.xml.sax`
  - JAXP (Java API for XML Parsing)
  - All about XML and Java
  - Provides three kinds of XML parsers (and XML-related technologies like Xpath):
    - DOM
    - SAX
    - StAX

Working with XML

- DOM
  - Document Object Model
  - Builds a full representation of the XML document in memory (implementations usually relies on SAX to build it), allowing for random access
- SAX
  - Simple API for XML
  - Push-like
  - Event-driven API (`startDocument()`, `startElement()`, `endElement()`, etc.)

Working with XML

- StAX
  - Streaming API for XML
  - Pull-like
  - Mix between DOM and SAX
  - Allows iterating over an XML document using `next()` and `hasNext()`

A little bit of StAX

```
XMLEventReader reader = XMLInputFactory.newInstance()
  .createXMLEventReader(new FileReader("build.xml"));
while(reader.hasNext()) {
  XMLEvent evt = reader.nextEvent();
  switch (evt.getEventType()) {
    case XMLEvent.START_ELEMENT:
      StartElement se = evt.asStartElement();
      if(se.getName().getLocalPart().equals("target")) {
        Attribute targetName =
          se.getAttributeByName(new QName("name"));
        System.out.println(targetName.getValue());
      }
      break;
  }
}
```

Web services

- `javax.xml.ws`
  - JAX-WS (Java API for XML Web Services)
  - Allows building applications that expose or consume Web services

OPEN SOURCE LIBRARIES FOR JAVA

Open Source libraries

- Warnings
  - The following list is clearly not exhaustive!
  - When coding in Java, Google (or Bing, or whatever search engine you enjoy) is your best friend to:
    - Search for existing libraries
    - Search for existing code
  - Avoid GPL-licensed libraries, use (worst case) LGPL-licensed libraries

Open Source libraries

- Commons BeanUtils
  - http://commons.apache.org/beanutils/
  - Wraps the Reflection API to ease working with JavaBeans
  - Main classes:
    - `BeanUtils`: Static methods for populating JavaBeans
    - `MethodUtils`: Static methods for working with methods
    - `PropertyUtils`: Static methods for working with getters/setters

Open Source libraries

- Commons Codec
  - http://commons.apache.org/codec/
  - Provides some encoders/decoders implementations (Base64, URLs, etc.)
  - Entry points:
    - `Base64`: Methods for working with base64
    - `Hex`: Methods for working with hexadecimal
    - `DigestUtils`: Methods for computing MD5 and SHA

Open Source libraries

- Commons Collections
  - http://commons.apache.org/collections/
  - Tons of enhanced classes for working with collections
  - Some classes:
    - `CollectionUtils`
      - `filter()`, `find()`, `transform()`, etc.
    - `LazyList`
    - `LazyMap`

Open Source libraries

- Commons IO
  - http://commons.apache.org/io/
  - Utility classes for performing I/O operations such as reading/writing files
  - Some classes:
    - `FileUtils`
      - `copyFile()`, `moveDirectory()`, `touch()`, etc.
    - `IOUtils`
      - `closeQuietly()`, `copy()`, `readLines()`, etc.
    - `DirectoryWalker`: Allows walking through a directory

Open Source libraries

- Commons Lang
  - http://commons.apache.org/lang/
  - Enhances the `java.lang` package by providing classes such as `DateUtils`, `StrTokenizer`, `StringEscapeUtils`, `StringUtils`
  - Some methods you MUST use:
    - `StringUtils.is[Not]Blank()`
    - `StringUtils.is[Not]Empty()`

Open Source libraries

- Commons Net
  - http://commons.apache.org/net/
  - Brings the support of many protocols such as Telnet, POP3, SMTP

Open Source libraries

- Derby
  - http://db.apache.org/derby/
  - Fully Java-based RDBMS
  - SQL- and JDBC-compliant
  - Can be embedded into applications
- HttpClient
  - http://hc.apache.org/
  - Implements the client side of HTTP standards

Open Source libraries

- iText
  - http://itextpdf.com/
  - Generates PDF files in Java applications
  - Used, for example, by Eclipse BIRT
- JFreeChart
  - http://www.jfree.org/jfreechart/
  - Generates charts in Java applications

Open Source libraries

- Joda-Time
  - http://joda-time.sourceforge.net/
  - Tons of classes to ease working with `Date`s (because working with `java.util.Date`s and `java.util.Calendar`s can quickly become a nightmare)
  - Served as the basis for JSR 310 (Date and Time API)

Open Source libraries

- OW2 ASM
  - http://asm.ow2.org/
  - Award-winning Java bytecode manipulation and analysis framework
  - Used by Eclipse, Groovy, Oracle, etc.
- POI
  - http://poi.apache.org/
  - Helps dealing with Microsoft Office files