Scientific Programming Using Object-Oriented Languages
Module 7: Building Larger Programs

Aims of Module 7:

- Gain overview of Java
  - how it works
  - why it is the way it is.
- Learn more about software development:
  - especially important for larger projects…
  - … but also applicable to small ones.

- No exercises for this module…
  - …but you will have to apply it in module 8…
  - …and in the final exam.

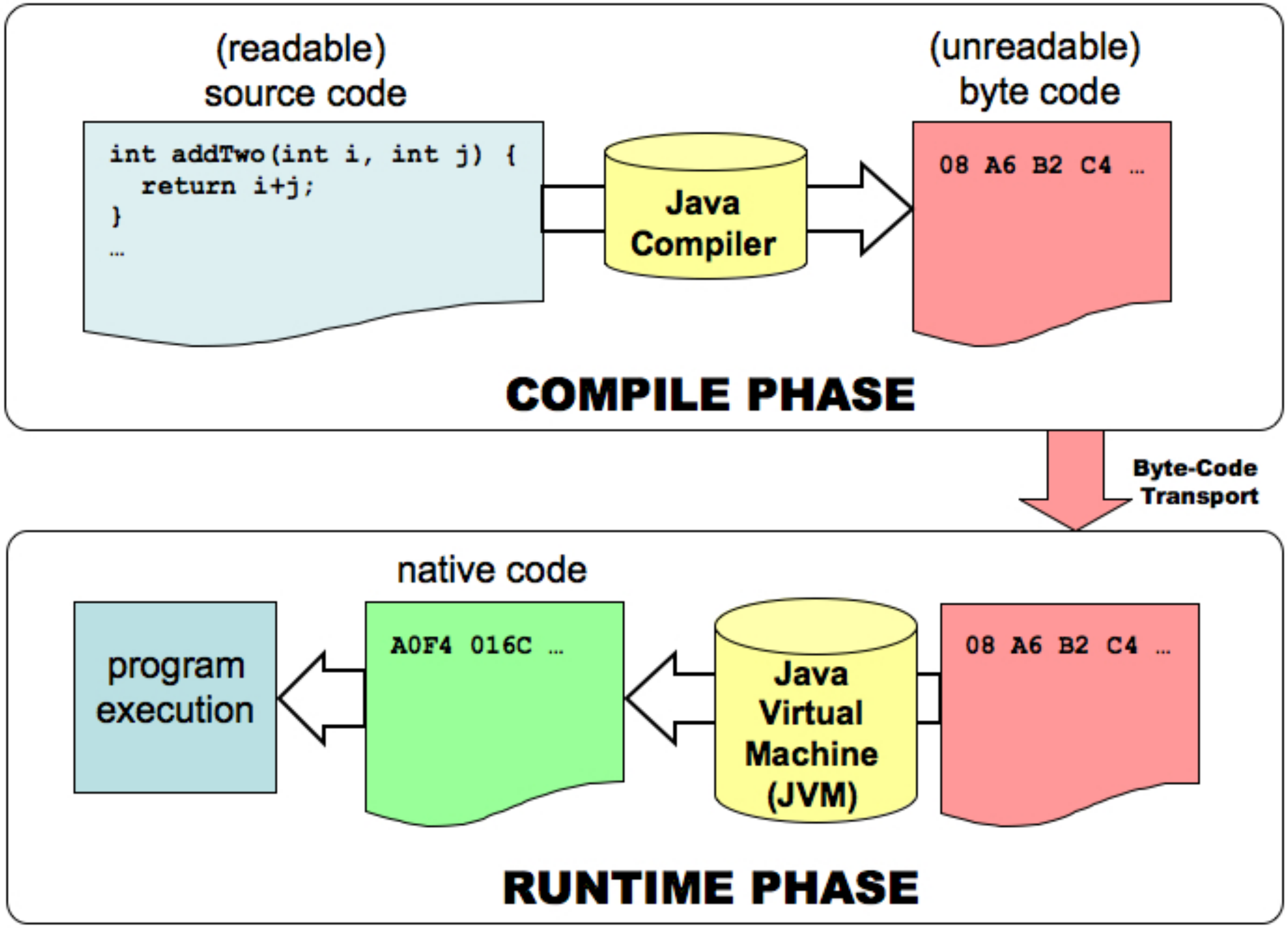
Java: History

- “Oak” language created by James Gosling as a by-product of another project at Sun in the early 1990s.
- Adapted to run in web browsers, enabling programs and animations to be embedded in web pages.
- Version history:
  - Java 1.0 released in January 1996 with the slogan “Write Once, Run Anywhere”: Java Runtime Environment provided for popular platforms.
  - Java 1.2 (December 1998) introduced major changes, later rebranded “Java 2”.
  - …
  - Java 5.0, aka 1.5 (September 2004): introduced generics, enhanced for loop…
  - …
  - Java 7 (July 2011): used for this course.
  - Java 8 (March 2014): improved support for functional programming

Java: How It Works

- Compiled languages:
  - source code is compiled into platform-specific machine code
  - fast execution
  - need to recompile if code changes
- Interpreted languages:
  - source code is interpreted as the program runs
  - can run on many platforms if interpreter is available
  - no need to recompile when code changes: faster development
- Java:
  - compile source code to “byte code”
  - byte code is interpreted by Java Virtual Machine (JVM)
  - JVM available for many platforms: “write once, run anywhere”
  - Speed:
    - originally quite slow
    - now can compete with compiled programs in many cases

Java: How It Works



Software Development

- Developing large, complex software is difficult and error-prone!
- Various approaches have been tried to make software development more reliable: *software engineering*.
- Range from very strict to informal *processes* or *methodologies*:
  - waterfall model
  - “unified process”
  - agile development.
  - test-driven development
- Key is compatibility with limited human working memory and attention span
  - work with small unit at a time
  - work with large units at higher level of abstraction

Software development

- Common themes in software engineering
  - structure: modularity, encapsulation
  - abstraction: high-level concepts hide complexity of lower levels
  - objects
  - testing
  - patterns: apply same “type of solution” in many different applications
  - e.g. “decorator” pattern in Java i/o streams: add functionality (buffering…) to underlying object instead of creating many new classes

Software Development

- Think about the problem:
  - What classes will you need?
  - Rough plan of structure: use pen and paper!
  - Don’t do too much detailed planning before starting to write code…
- Iterative/incremental development:
  - Start with a simple program that works, not a complex one that doesn’t!
  - Make small changes and test at each stage.
  - May be necessary to rethink design along the way.
- Clarity
  - Good code is readable!
  - If it is not clear what your code is doing, it is unlikely to be doing what you want it to.

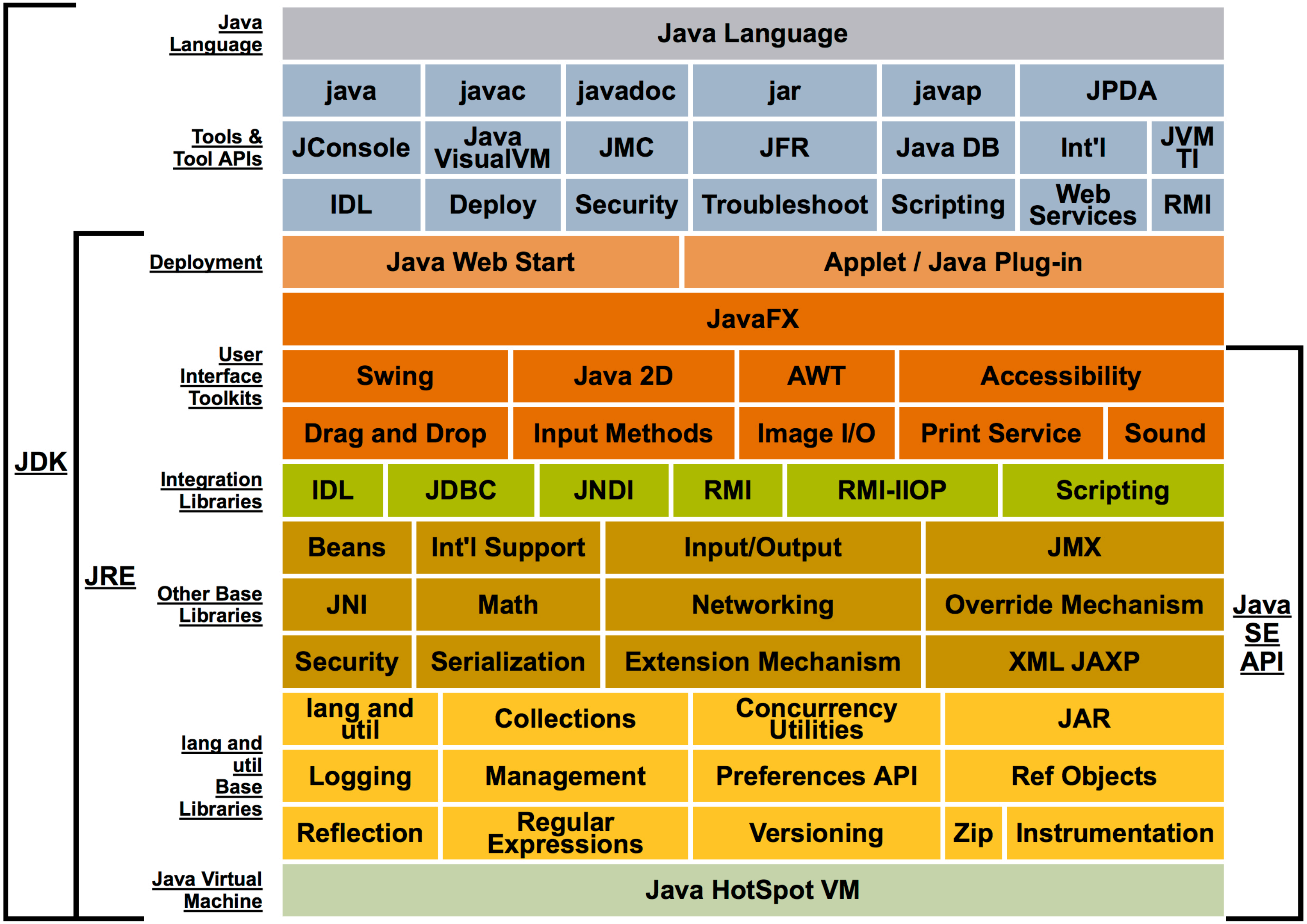
Java: Type Checking

- Java is a *statically typed* language.
- The compiler knows what type of value each variable refers to.
- Type compatibility can be checked at compile time, before trying to run the program...
- ...or even earlier, as you type in an IDE.
- Catches many potential bugs early on
  - Can't accidentally treat a `double` as an `int`
  - ...or a `File` as a `PrintWriter`

Java: Packages

- Large projects can have many classes
  - nearly 3000 classes and interfaces listed in Java API documentation
- Group these into “packages” for manageability
  - classes that are related and depend on each other are grouped together.
  - can also have subpackages.
- Only import the classes/packages you need
  - avoid name clashes…
  - classes in the same package, and in `java.lang`, are imported automatically.
  - use fully qualified name to resolve ambiguities: e.g. `java.util.Vector` vs `phas3459.test.Vector`
- A package corresponds to a directory/folder
  - source code for class `uk.ac.ucl.hep.Positron` must be in a file …\uk\ac\ucl\hep\Positron
-

Java: The API



JDK

JRE

Java SE API

Java Language
Java Language

Tools & Tool APIs
java, javac, javadoc, jar, javap, JPDA
JConsole, Java VisualVM, JMC, JFR, Java DB, Int'l, JVM TI
IDL, Deploy, Security, Troubleshoot, Scripting, Web Services, RMI

Deployment
Java Web Start, Applet / Java Plug-in

JavaFX

User Interface Toolkits
Swing, Java 2D, AWT, Accessibility
Drag and Drop, Input Methods, Image I/O, Print Service, Sound

Integration Libraries
IDL, JDBC, JNDI, RMI, RMI-IIOP, Scripting

Other Base Libraries
Beans, Int'l Support, Input/Output, JMX
JNI, Math, Networking, Override Mechanism
Security, Serialization, Extension Mechanism, XML JAXP

lang and util Base Libraries
lang and util, Collections, Concurrency Utilities, JAR
Logging, Management, Preferences API, Ref Objects
Reflection, Regular Expressions, Versioning, Zip, Instrumentation

Java Virtual Machine
Java HotSpot VM

Java: Testing Code

- Why test your code?
  - If you haven't tested your code, how do you know it works?
  - Will it still work if you make a minor change?
  - Will it work on all valid input data?
- How to test code
  - compile-time checks
  - run it and check output
  - run it with different input
  - but can't test all possible inputs, and some parts of the code may not be tested properly
- Unit testing
  - write code in small *units* (classes, methods) that are easier to test
  - write dedicated *tests* that verify the behaviour of these units

Java: Testing Code

- Java testing frameworks
  - *automated* testing
  - JUnit
  - TestNG
  - not just unit tests
- Writing JUnit tests
  - see demo

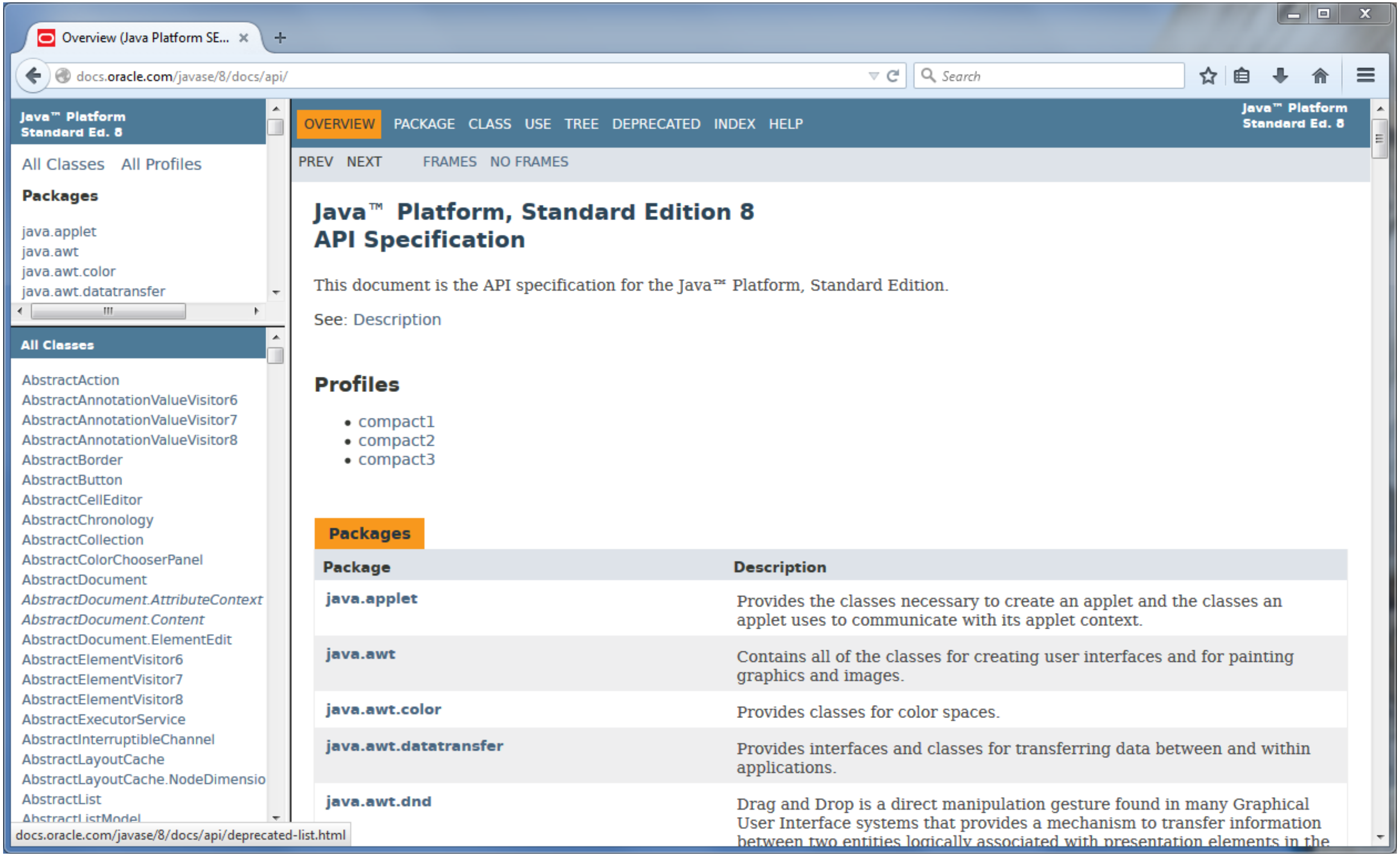
Java: Documenting Code

- Specially formatted comments processed by Javadoc to create web pages.
- Enclose Javadoc comments in `/** … */`
- Insert comment immediately before class, method or member variable
- Use the following tags:
  - for classes:
    - `@author`
    - `@version`
  - for methods:
    - `@param`
    - `@return`
    - `@throws`
- Explain meaning of parameters, return values and exceptions

```
/** MyClass is …
 *
 * @author Frank Deppisch
 * @version 1.0
 */
public class MyClass {

  /** myMethod does …
   *
   * @param a  Number of …
   * @return   Gives the …
   */
  public int myMethod(int a) {
     // …
  }
}
```

Java: Documenting Code


Overview (Java Platform SE...
docs.oracle.com/javase/8/docs/api/
Search
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV NEXT FRAMES NO FRAMES
Java™ Platform Standard Ed. 8
docs.oracle.com/javase/8/docs/api/deprecated-list.html

Java™ Platform
Standard Ed. 8

All Classes All Profiles

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensio
AbstractList
AbstractListModel

Java™ Platform, Standard Edition 8 API Specification

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

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the