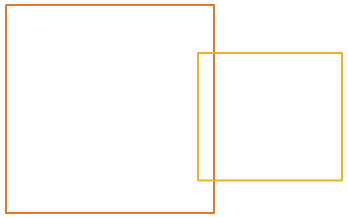
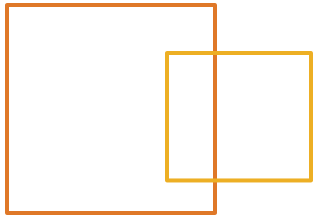
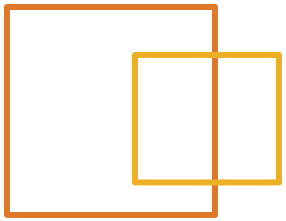
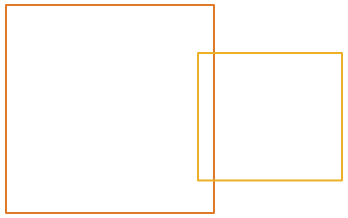
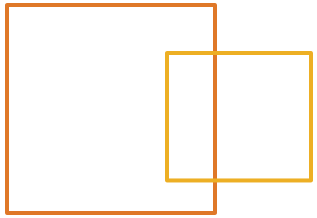
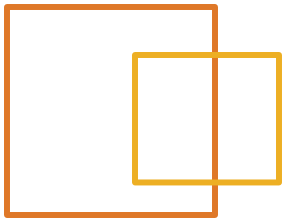





Fast Track to Java

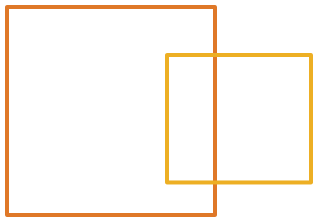
Customized for Starbucks

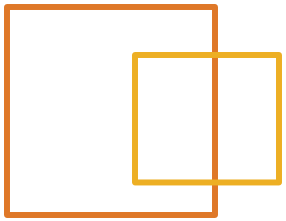
Delivered by DevelopIntelligence


Review of the Java Platform

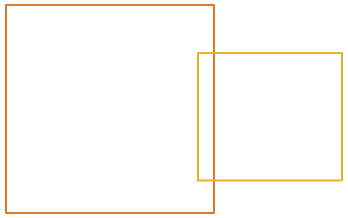
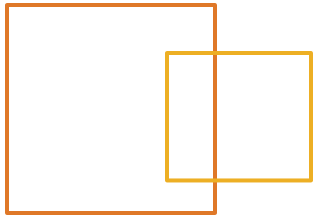
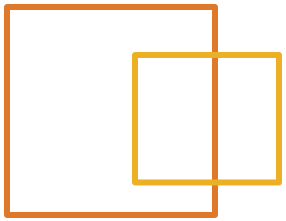
Objectives





At the end of this module you should be able to:

- Understand Java
- Discuss why Java should be used and who owns it
- Talk about Java Classifications
- Use SDK & JRE
- Create Jar files
- Use the Java Programming Model
- Write, compile, and run Java Applications

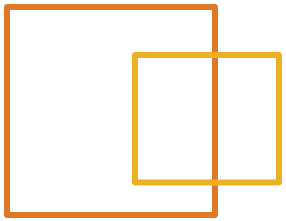

History of Java

Green is in!

Just a Seed

- Java started out as a research project
  - Research began in 1991 as the Green Project
  - Project was chartered to anticipate and plan for next wave of computing
  - “Green Team” determined consumer devices and computers would converge
  - Team focused on TV set-top boxes and interactive TV industries

Only a Sapling

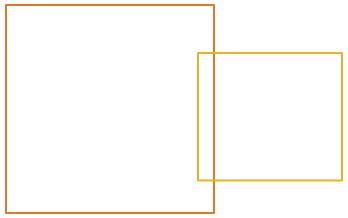
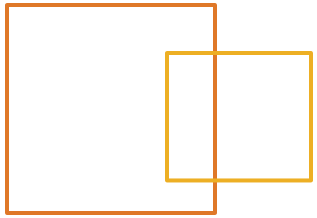
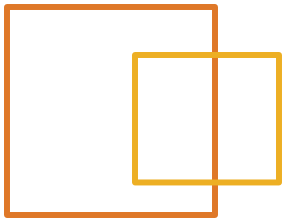


- Research efforts birthed a new language, OAK
  - Oak was renamed Java in 1994
  - Created by James Gosling - *"the father of Java"*

- Language was created with 5 main goals:
  - It should be object oriented
  - A single representation of a program could be executed on multiple operating systems
  - It should fully support network programming
  - It should execute code from remote sources securely
  - It should be easy to use

The Budding Tree

- Java was publicly released May 27, 1995
  - As a product, it was targeted at Internet development
  - In general, it was marketed as the language to add dynamic features to the web, a.k.a. Applets
  - Had early support from companies like Netscape Communications

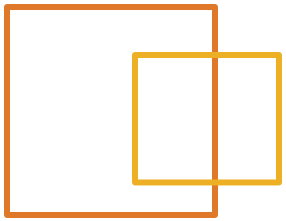





What is Java?

What is Java?

- Java is defined by two entities:
  - A platform (Java Runtime Environment - JRE)
  - A language (Java Software Development Kit - SDK)
- Java address less traditional concerns like
  - Security
  - Reusability
  - Transportability (platform independence)
  - Network capability
- Created by Sun Microsystems, now guided by Oracle

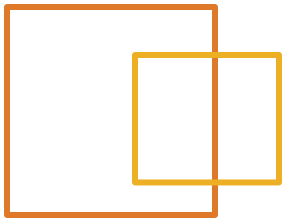
The Java Platform



- The Java platform provides
  - The run-time environment
  - The necessary libraries (platform libraries)

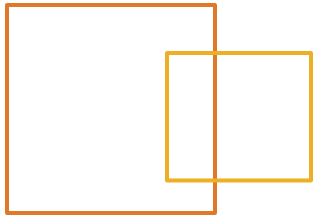
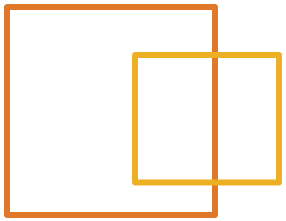
- The Java platform software is NOT platform independent
  - Platform implementations exist for almost every Operating System
  - Windows, OS X, Linux, Solaris, AIX, HP-UX, VMS, OS/2, OS/400, many embedded systems, and many more

Platform Editions



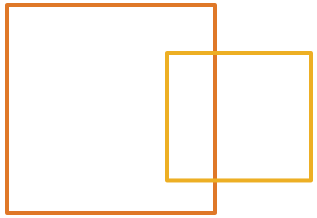
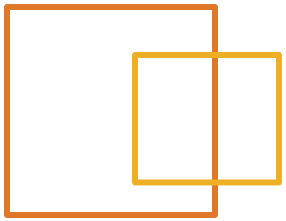
- Java has different platform editions
  - Java Standard Edition (Java SE)
  - Java Enterprise Edition (Java EE)
  - Java Micro Edition (Java ME)

- Editions defined in terms of JVM and platform libraries
  - Each platform has its own set of “libraries”
  - All editions rely on a Java Runtime Environment and Java Virtual Machine

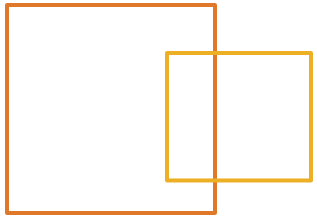
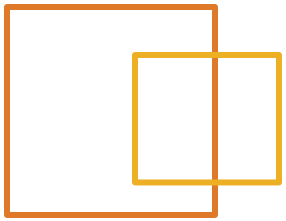




- Complete environment for application execution
  - Stand-alone server applications
  - Stand-alone client applications
  - Stand-alone client-server applications
  - Applets
  - Mobile applications
  - Web-start applications - rich applications deployed via Web

- Considered ‘core’ to all editions

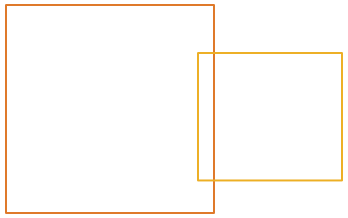
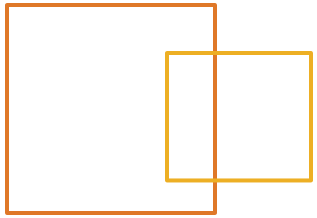
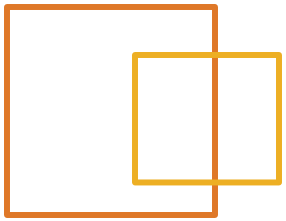




- Extension of Java SE - uses Java SE run-time environment
  - Adds libraries, frameworks, and container concept
- Targeted at enterprise applications; applications that span all areas of an enterprise
  - From customer to back-office
  - From web to legacy
- Enables distributed multi-tier solutions

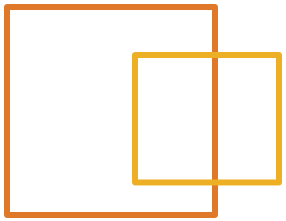

- Targeted at consumer and embedded market; constrained devices

- Defined variations for major categories of device physical capability
  - Connected Device Configuration (CDC)
  - Connected Limited Device Configuration (CLDC)

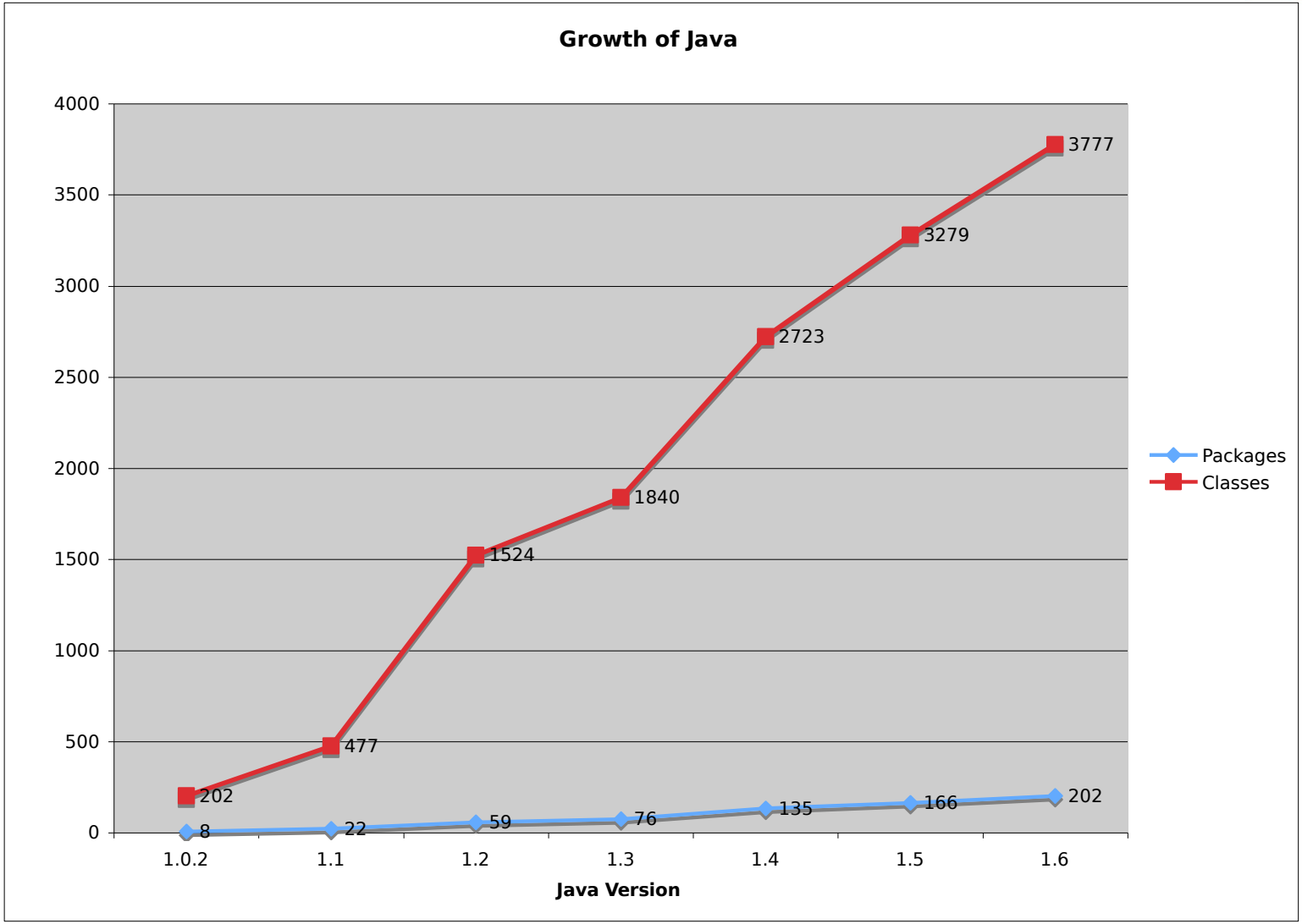

Current State of Java

A Growing Forest

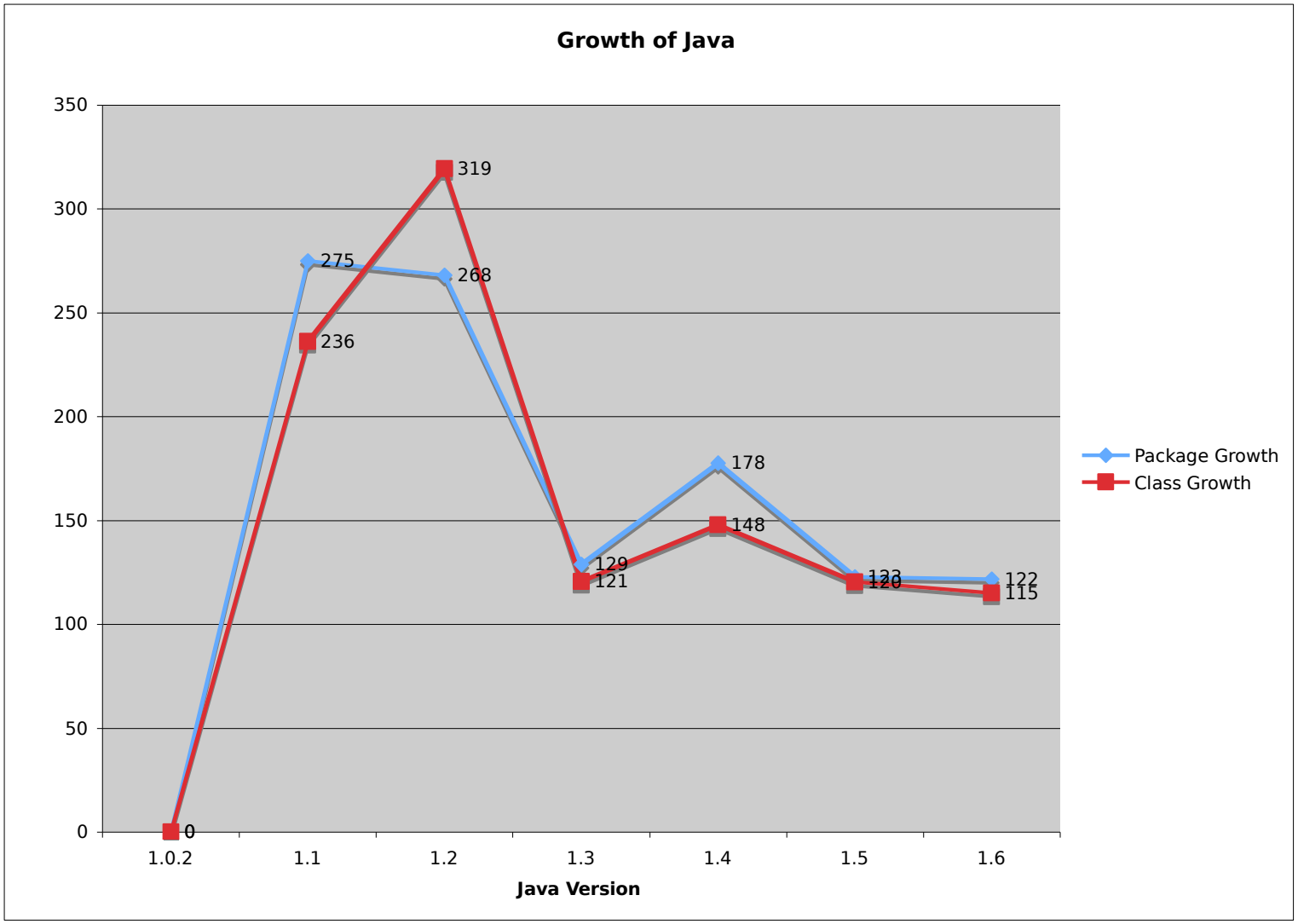


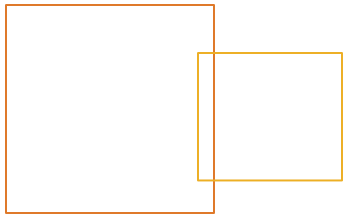
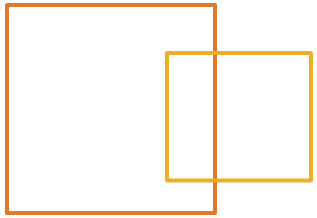
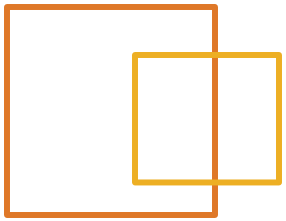
- First released in 1995
- Current version of Java is Java 7

Breadth of Java Across Versions



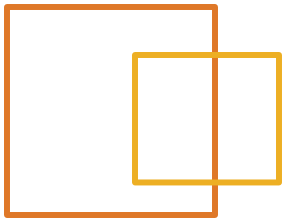
Growth of Java Across Versions




Java Standard Edition

Java SE Platform



- Represents the “historic” Java platform
- Considered the “core” Java platform
  - Used for browser plug-ins to stand-alone Java applications
  - Extended to support enterprise application development (Java EE)
  - Constrained to support micro application development (Java ME)
- Typically discussed in terms of its:
  - Runtime environment (JRE)
  - Development environment (JDK)

Java SE Platform

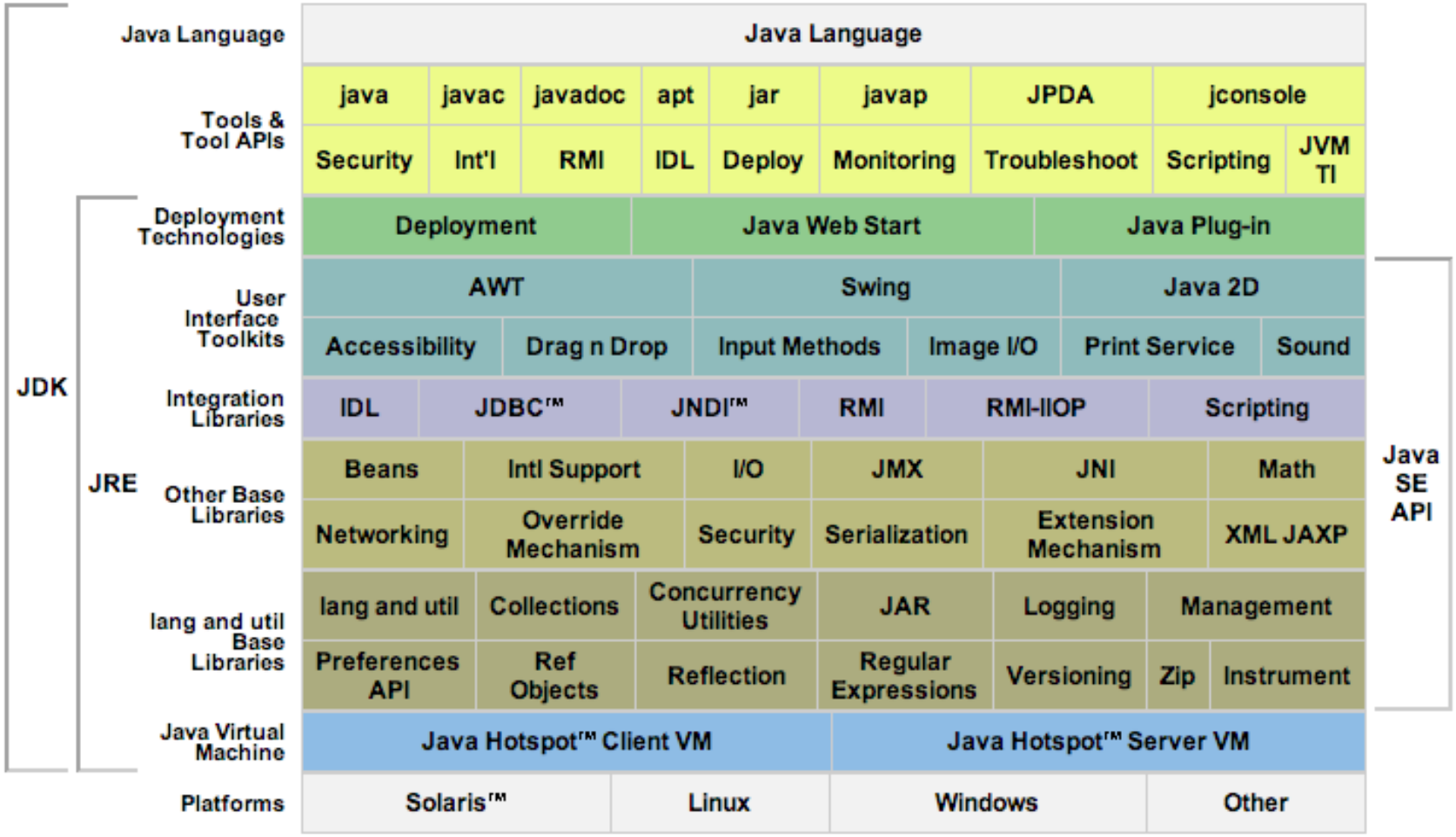


Image location: http://java.sun.com/javase/6/docs/

Java SE Runtime Environment

- Considered the execution platform
- Consists of two primary facilities
  - Java Virtual Machine (JVM)
  - Java SE Application Programming Interfaces (API)
- When put together, they are considered the JRE
  - JRE implementation is operating system specific
  - JRE has consistent behaviors and capabilities across operating systems
  - JRE is the only necessary piece required to run a Java application

Java SE Platform [JVM]

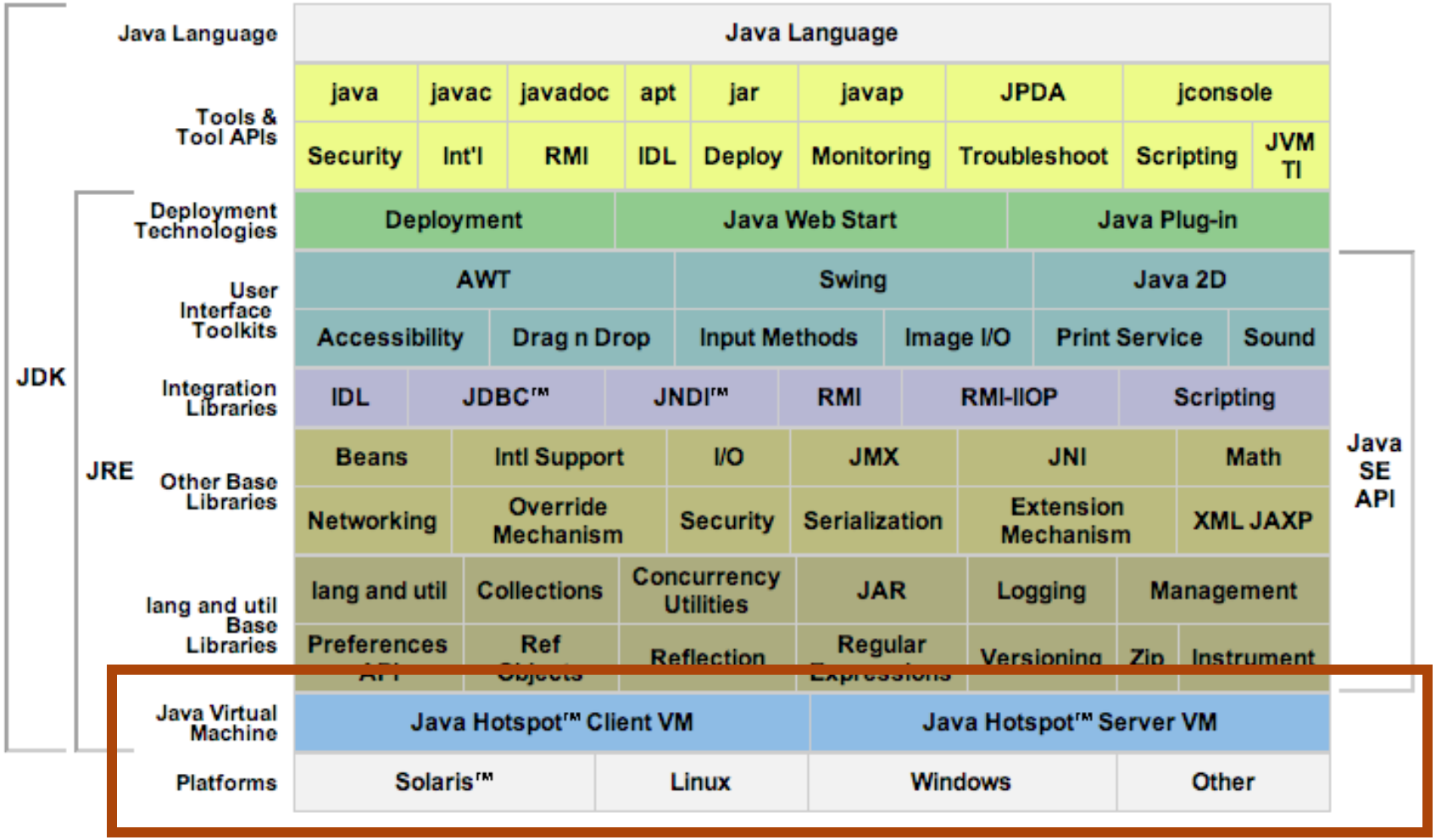


Image location: http://java.sun.com/javase/6/docs/

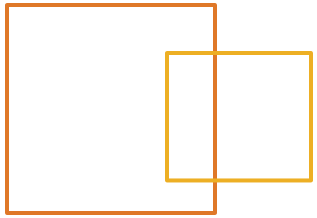
Java Virtual Machine

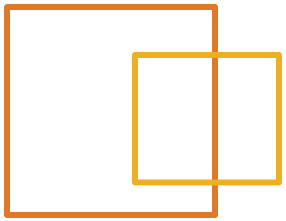
- Stand-alone OS native application
  - Executes bytecode
  - Bytecode represents compiled Java source code
  - JVM is operating system dependent
- JVM acts as facilitator between a Java application and the OS
  - Isolates application from OS quirks
  - Provides consistency across OS to application
  - Contains a host of rich execution facilities
- Specification driven

Java Virtual Machine [cont.]

- JVM variations
  - *Interpreters* – first form of Java: pure interpretation
  - *Compilers* - native code; fast, but platform dependent binary
  - *Just-In-Time Compilers* (JIT) – Current form of Java: partial interpretation, partial compilation at runtime
- JVM Implementations
  - JVM functionality is defined by a specification
  - Different vendors have different implementations
  - May have different execution modes and tuning characteristics

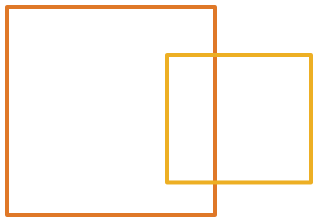
JVM Facilities

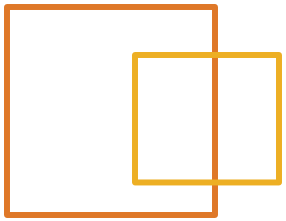




- JVM provides:
  - Platform independent execution
  - Dynamic binding
  - Thread management
  - Automatic memory management
  - Security model

Java SE API





- Typically discussed in terms of libraries (APIs)
  - APIs are presented as “packages”
  - Each ***`package`*** is a logical grouping of related functionality
- Libraries broken in four categories:
  - Language
  - Base
  - Integration
  - UI toolkits
- Platform may be “extended” through additional packages
- Community driven

Java SE Platform [API]

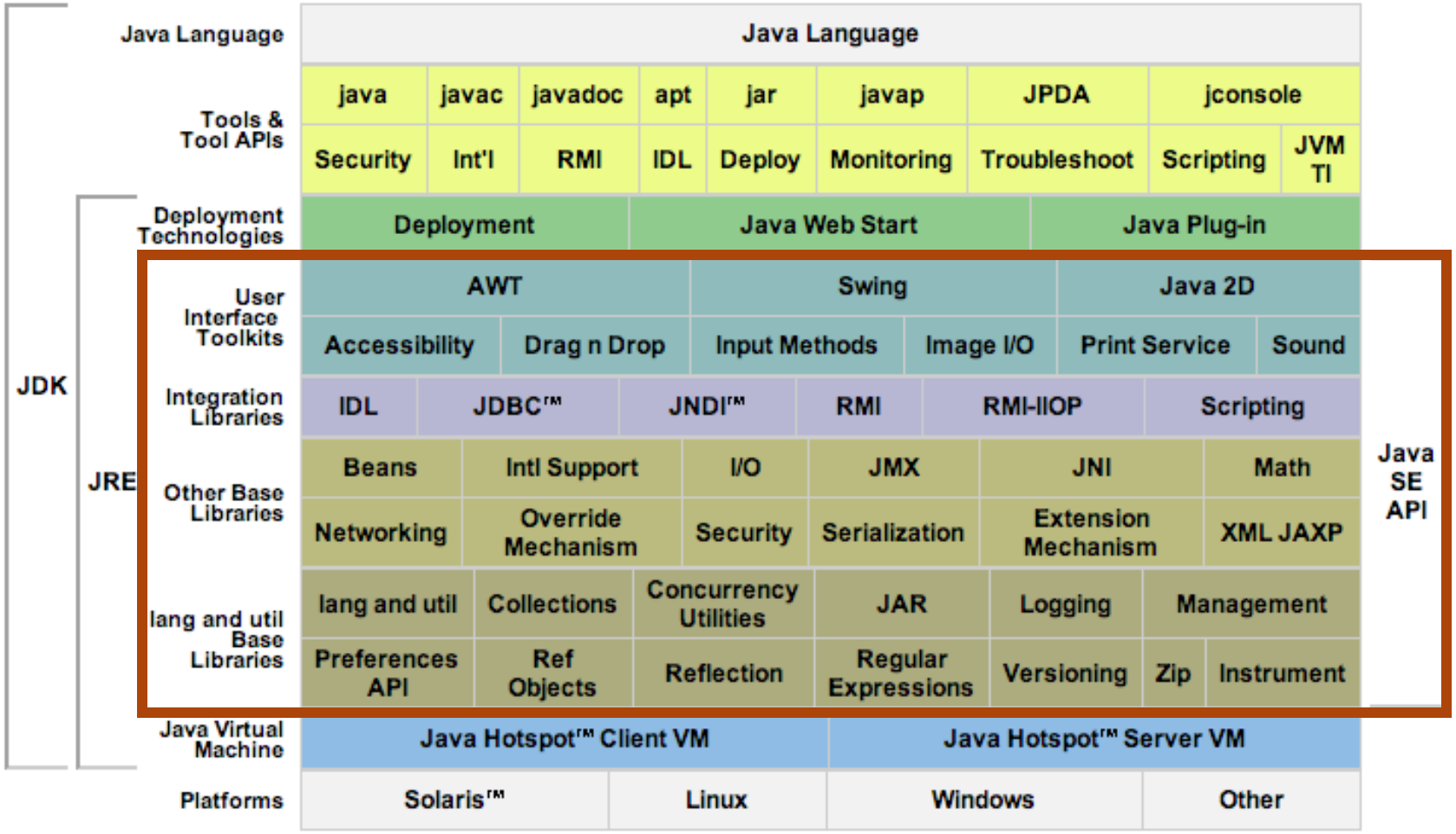
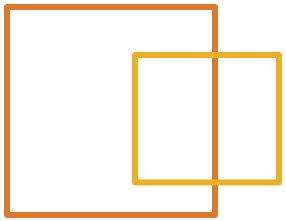


Image location: http://java.sun.com/javase/6/docs/

Java Language Packages

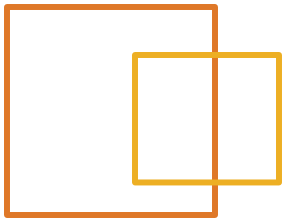
- Provide implementation of language characteristics and functionality
- Governed by the Java Language Specification
- Available in all “platforms”
- Typically found in ***`java.lang`*** packages like:
  - ***`java.lang`***
  - ***`java.lang.annotation`***
  - ***`java.lang.ref`***
  - ***`java.lang.reflect`***

Base Packages



- Considered foundational to the SE platform
- Normally found in ***`java.`*** package structure, like:
  - ***`java.io`***
  - ***`java.net`***
  - ***`java.util`***
- Subset of base packages found in Java ME

Base: Input / Output



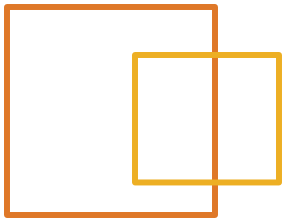
- Provides platform independent I/O mechanism
  - Supported by abstraction of file system
  - OS specifics handled by native implementation

- Two types of I/O
  - Synchronous I/O - ***`java.io`***
    - Follows stream-based model
    - Supports text and “binary”
  - High performance I/O - ***`java.nio`***
    - Follows channel-based model
    - Asynchronous
    - Supports buffers

Base: Network Programming

- Rich support for networked applications
  - Found in ***`java.net`***
  - Underlying communication handled by OS
- Supports transport layer communication
  - TCP - sockets and server sockets
  - UDP - packets and sockets
- Support for application-layer programming
  - Through ***`java.net.URL`*** and ***`java.net.URLConnection`***
  - Includes support for things like:
    - Http
    - Mailto
    - FTP

Base: Data Structures



- Collections API provides standard data structures
- Lists, Sets, Maps, Queues, etc.
- Thread-safe and non-thread-safe implementations
- Built-in and extensible sorting, selection, and ordering facilities

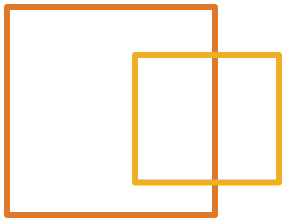
Integration Packages

- Contain libraries and functionality to integrate with other systems
- Implemented using a layered approach
  - Provide a Write-Once-Run-Anywhere integration capabilities
  - Abstract the application from system specifics
- Includes things like:
  - ***`java.sql`***
  - ***`java.rmi`***
  - ***`javax.xml`***

Integration: Database Programming

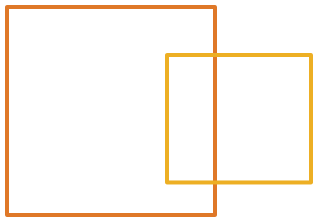
- Java Database Connectivity (JDBC)
  - WORA for databases
  - JDBC provides a set of database independent APIs
  - DB specific interactions provided by JDBC-compliant driver
  - Supports connections to multiple databases at a given time
- Found in two packages:
  - ***`java.sql`***
  - ***`javax.sql`***
- Capabilities leveraged by Java EE

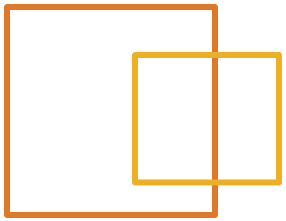
Integration: RMI



- Stands for remote method invocation
- Java specific distributed computing mechanism
- Introduced in JDK 1.1
- Built into the Java platform - ***`java.rmi`***
- Distributed computing platform for:
  - Distributed Object-oriented computing
  - Enterprise Java Beans
  - Jini

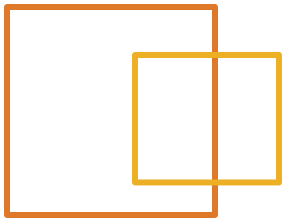
UI Toolkits





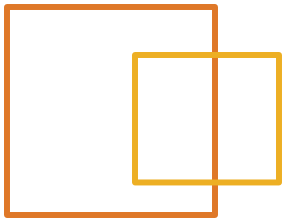
- User Interface development supported through:
  - Abstract Windowing Toolkit
    - Basic; least-common-denominator widget set
    - Relies on *native-peers*
    - Platform-specific look and feel
  - Swing / Java Foundation Classes
    - Advanced; full-featured
    - Written in Java
    - Pluggable look and feel with option for consistent, platform-independent, look and feel
- Provides WORA for graphical-based applications

Platform Extensions



- Considered extensions to the platform
  - Not necessarily considered “core” facility of platform
  - Typically governed by specification falling outside “platform specification”
  - Usually bundled with platform, but could be third-party
- Typically have a ***`javax`*** package structure like:
  - ***`javax.naming`***
  - ***`javax.swing`***
  - ***`javax.transaction`***
- Many javax packages are now in the core
  - Inconvenient to rename once released & in regular use

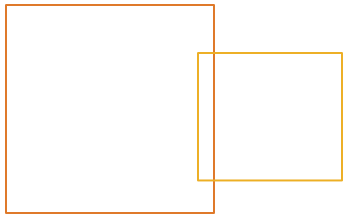
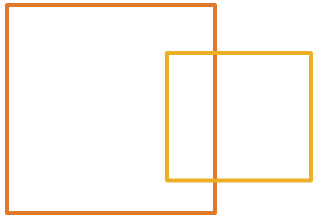
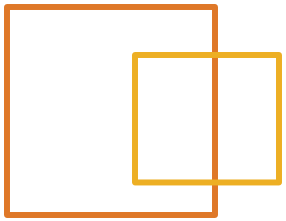
Types of Java Applications



- Java SE is targeted at creating “classic” applications
  - Few constraints on class; any class can be an application
  - JVM executes a lifecycle method
  - Lifecycle method must have specific signature:
    ```
    public static void main(String [] args)
    { … }
    ```
- Classic applications include:
  - Standalone client
  - Client-server
  - Server
  - Distributed (peer-to-peer)

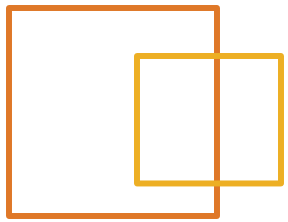
Types of Java Applications [cont.]

- Java SE also supports some web-based application approaches
- Two types of web-based applications
  - Applets
  - Web-start applications
- Both have similar characteristics in terms of:
  - Deployment
  - Security
  - Execution


Java SE Development

Java SE Development



- Application development provided through Java Development Kit (JDK)
- JDK contains:
  - Java SE JRE
  - A set of development tools
- All development tools are command-line
- Rich development environment provided through Integrated Development Environments (IDEs)

Java SE Platform [JDK]

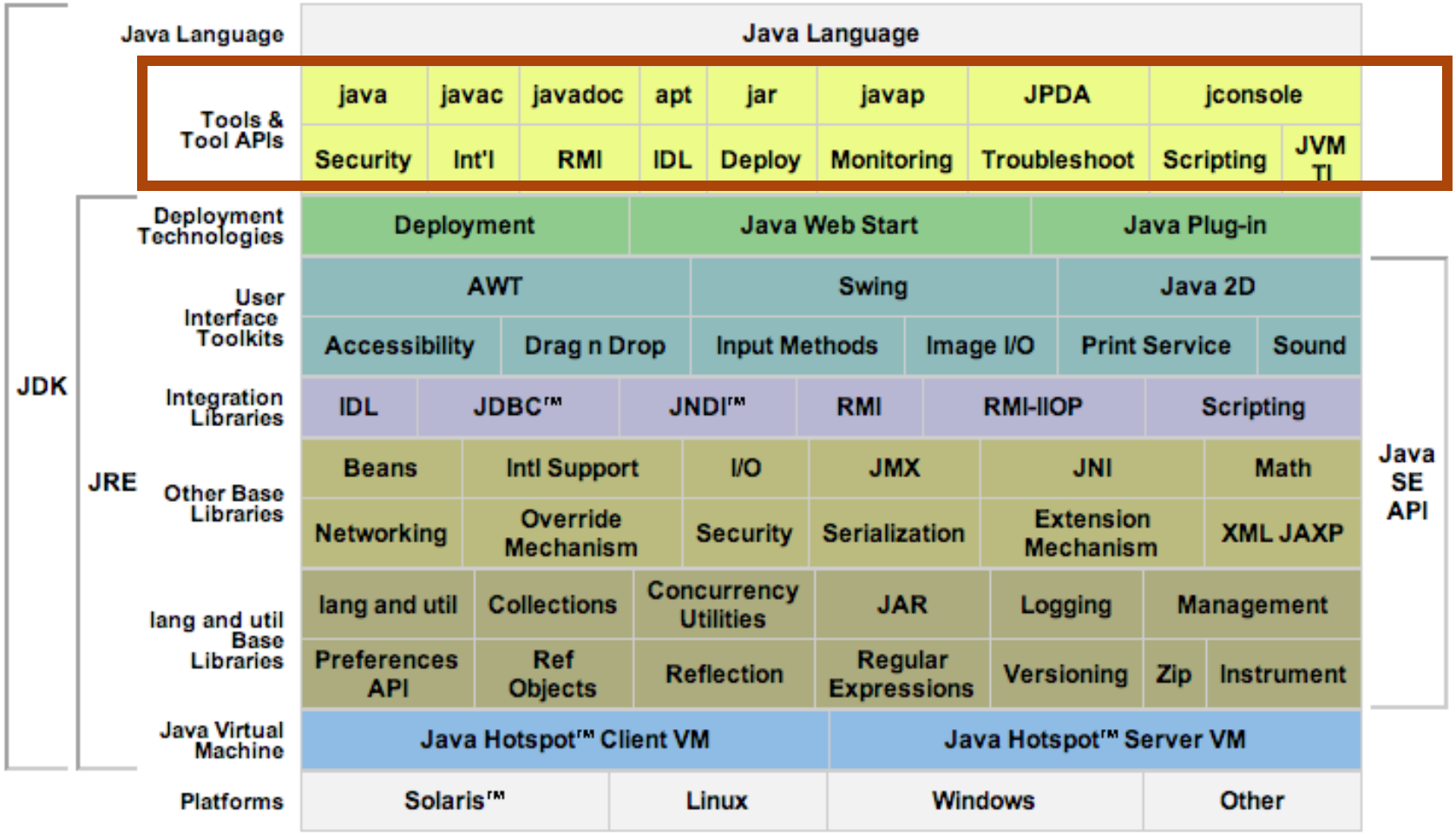


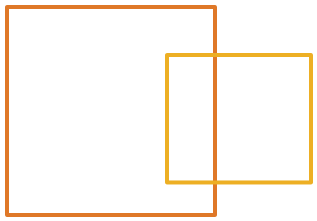
Image location: http://java.sun.com/javase/6/docs/

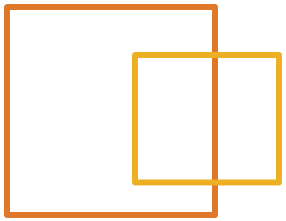
Core Java Development Concepts

Java language source code defined in text files:

- Source files provide:
  - Definition of entities and rules
  - Description of how entities interact
- Source files have basic requirements:
  - Filenames are case and white-space sensitive
  - File extension must be **`.java`**
- Source files become executable after compilation
  - Executable files contain bytecode
  - File extension is **`.class`**
  - At least one bytecode file generated per source file
  - Bytecode files are platform independent

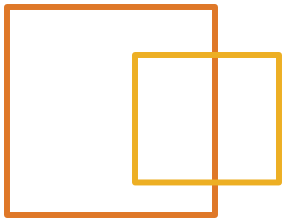
Java Compiler





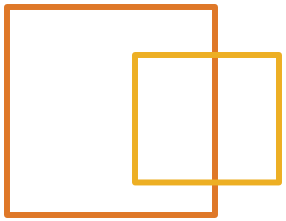
- Java compiler provided as part of JDK
- Written as Java program
  - Invoked on the command-line: ***`javac`***
  - Relies on a classpath
  - Supports cross-versioning compilation
- Uses multi-pass algorithm
  - Basic syntax checking
  - Type verification / validation checking
  - Exception handling
  - Identifies and notifies errors (and line numbers)
- Generates Java Virtual Machine compliant bytecode

Java Application Launcher



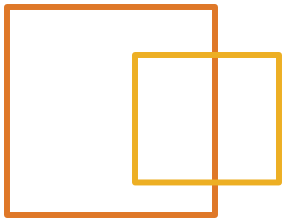
- Application launcher provided as part of JRE
- Used to “start” a stand-alone Java application
  - Invoked on the command-line: ***`java`***
  - Starts a Java Runtime Environment
  - Loads “platform libraries”
  - Loads and starts the application
- Has many “configurable” options
  - Classpath
  - Memory management algorithm
  - Memory size
  - Remote management

Other JDK Tools



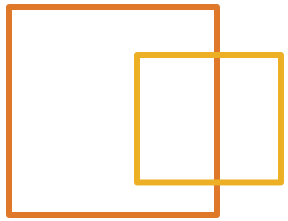
- Java Documentation Generator
  - Command-line documentation tool: ***`javadoc`***
  - Generates HTML-based documentation from source code
  - Useful for creating developer-oriented documentation
- Java Debugger
  - Command-line debugger: ***`jdb`***
- Third party IDEs combine editor, documentation, debugging, refactoring, version control, and other tools:
  - NetBeans (Sun/Oracle), Eclipse (IBM), and many others

Java Programming Model



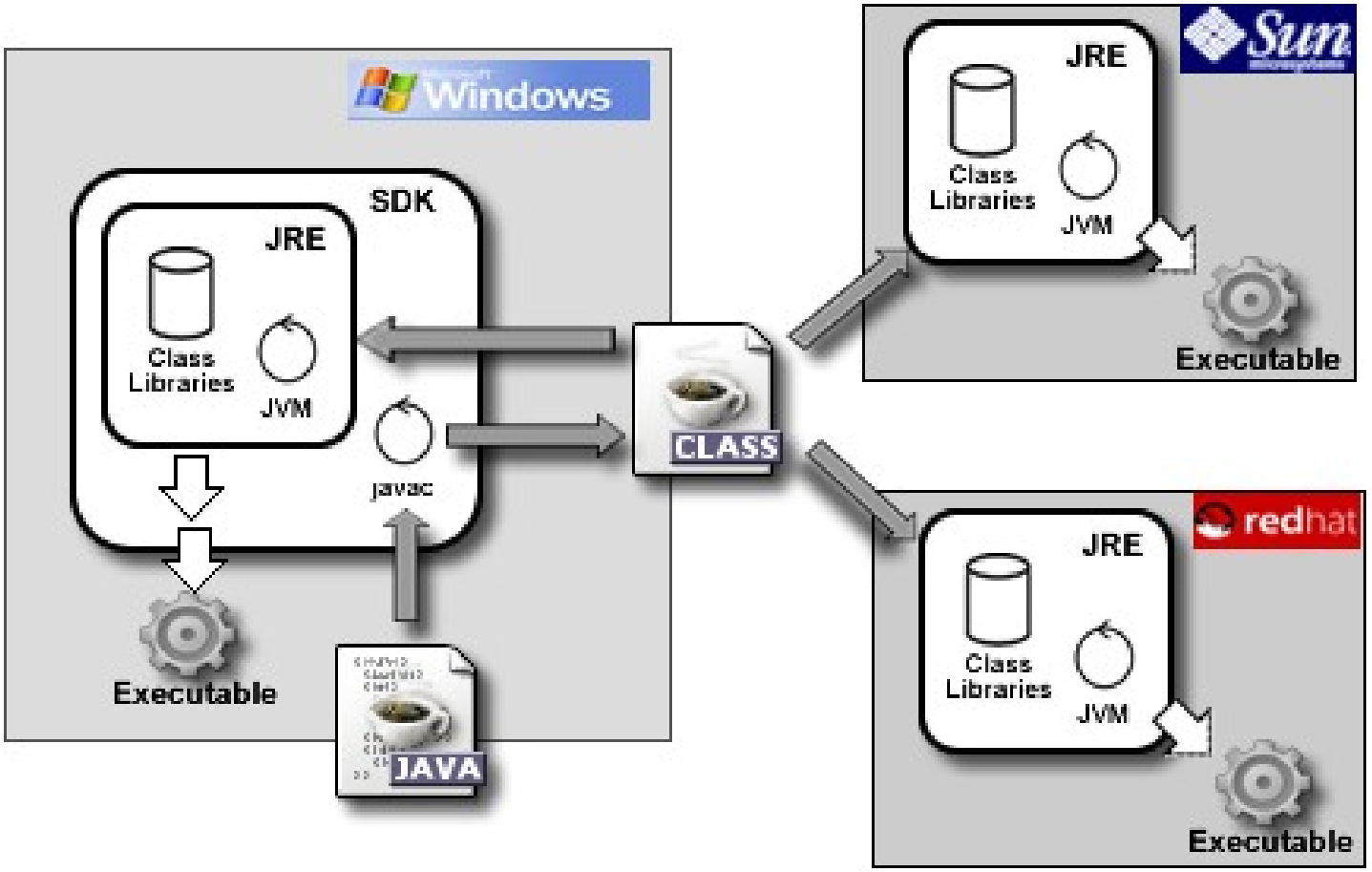
- Create source code
  - Stored as test file
  - Has extension .java
- Compile source code
  - Utilize java compiler - javac
  - Does syntax and language validation
  - Generates platform independent bytecode
  - Stored in a .class file

Java Programming Model

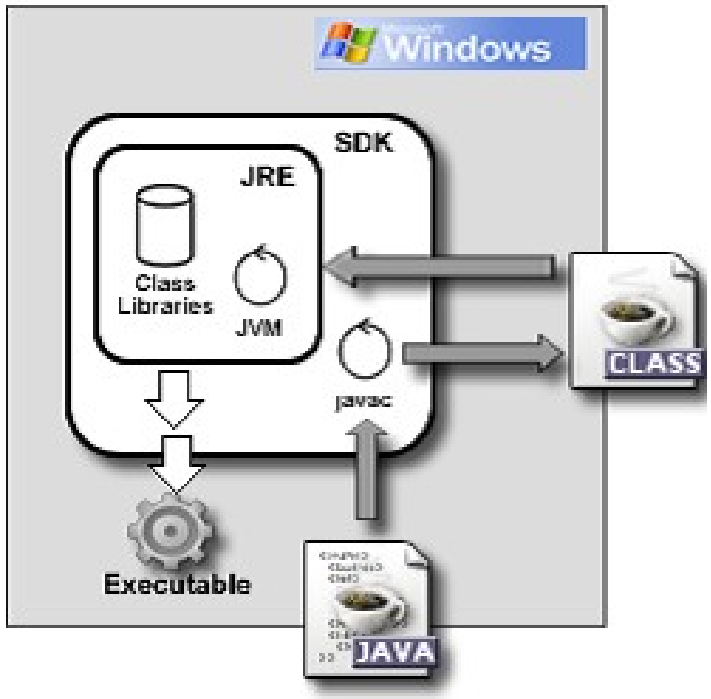


- Distribute .class files
  - On the web (for applets)
  - On the server (for enterprise applications)
  - On the client (for applications)
- Execute the “application”
  - Use the Java Runtime Environment (JRE)
  - JRE utilizes a Java Virtual Machine (JVM)
  - JVM loads class files and executes them
- Or let the IDE handle compilation/execution

Java Programming Model



Stand-alone Java Application



Compiling and Executing Java Applications

Developing

HelloWorld.java

```
public class HelloWorld {
  public static void main(String [] args) {
    System.out.println("Hello World!");
  }
}
```

```
> javac HelloWorld.java
```

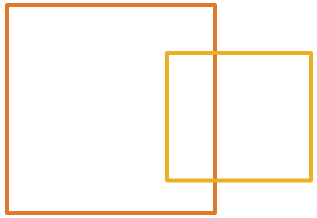
HelloWorld.class

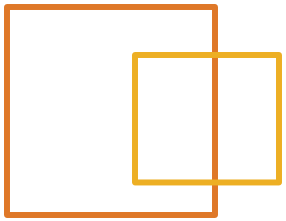
Executing

```
> java HelloWorld
```


Hello World!

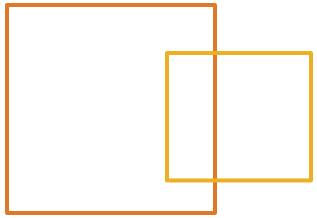
Summary

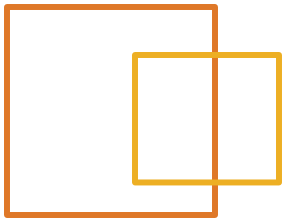




- We covered:
- Understand Java
- Discuss why Java should be used and who owns it
- Talk about Java Classifications
- Use SDK & JRE
- Create Jar files
- Use the Java Programming Model
- Write, compile, and run Java Applications

Lab 1





- Create a workspace for the course
- Write a “hello world” program to make sure everything is in place