Java for Adobe Experience Manager

Toby Dussek

Spring 2020

Welcome

- Introductions

Introductions

- What you know
- What you want to know

Starting AEM

- java –jar c:\adobe\AEM\author\aem-author-4502.jar
- First run can take a long time
- When web interface opens, log in (default is admin admin)

- AEM admin documentation
  - https://docs.adobe.com/content/help/en/experience-manager-65/administering/home.html
- Also see technical requirements for your version of AEM
  - https://docs.adobe.com/content/help/en/experience-manager-65/deploying/introduction/technical-requirements.html

AEM Stack

- Adobe Experience Manager is a Java web application based on the following standards
  - Apache Felix—an OSGi Framework
  - Apache Jackrabbit Oak
  - Apache Sling
  - HTML Template Language (HTL)
  - Apache Tika
  - Apache PDFBox

OSGi Bundles

- AEM allows bundles to be uploaded which contain the standard primitives for building re-useable collaborative components
- OSGi supports modular deployment of these bundles
  - Managed via AEM Web Console, Command Line options or Content Nodes in the JCR (preferred)
- Similar to Node Package Modules (npm) in JavaScript
- To listen to OSGi events you register @Service
  - org.osgi.service.event.EventHandler
  - with an event.topics property

OSGi Annotations

- Annotations from org.apache.felix.dependencymanager.annotation
  - Component
  - Activate
  - Deactivate
  - Modified
  - Service
  - Reference
  - Property

- Activate and Deactivate are lifecycle events

Granite

- Granite is the technical foundation on which Adobe Experience Manager is built
- Granite provides the following components
  - An application launcher
  - An OSGi framework into which everything is deployed
  - A number of OSGi compendium services to support building applications
  - A comprehensive Logging Framework providing various logging APIs
  - The JCR API specification based on Apache Jackrabbit
  - The Apache Sling web framework

Java Content Repository (JCR)

- JCR is a database that looks like a file system
- It is unstructured, and enables versioning and observation
- It provides services such as full-text search, indexing, access control, and event monitoring

Java

- Java 8 (aka 1.8)

Compiled Java

- A program is translated to machine code so the operating system can load and manage the process
- Java pre-compiles code to platform-independent byte-code
  - Java byte-codes give hardware independence
  - Must be translated when program runs
- This makes Java hardware-neutral
- Java code runs in a Java Virtual Machine (JVM)
  - Applications run unchanged on almost any platform
  - There are ports of the JVM to all major platforms
- The JVM translates byte-code into platform-specific machine instructions

Java Archive (JAR) files

- This is just a zip file
- Add a class file to an archive
  - jar -cf output.jar AnyClass.class
    - c means you want to create a JAR file
    - f means you want the output to go to a file (rather than STDOUT)
    - m means you want to include a manifest file
- Optionally check contents of an archive
  - jar -tf output.jar
- Execute code from a .jar file
  - java output.jar

Create and Run a Jar file

- First compile the Java classes
  - javac MyProg.java MyHelper.java
- Then create a JAR file containing the .class files and the manifest file
  - jar cfm MyFirstJarFile.jar Manifest.txt MyProg.class MyHelper.class
- Run java to launch the Java Virtual Machine
  - The JVM consults the manifest file to determine the main class
    - java -jar MyFirstJarFile.jar

Java Tools

- Run jps to see details of running Java code
- Run jvisualvm for a graphical Java tool

Java IDEs

- Visual Studio Code (free)
  - Very easy IDE for simple Java coding
- NetBeans IntelliJ
  - Industry-leading tool
- Eclipse
  - Venerable opensource IDE with very strong following
  - AEM tools for Eclipse
    - https://marketplace.eclipse.org/content/aem-developer-tools-eclipse
    - Drag-drop into Eclipse to install or install via Marketplace
    - Includes a perspective for AEM

Maven

- Generate a fresh project from a template
  - mvn archetype:generate
  - Provide some values as it generates the project and unit test skeletons
  - Also generates a Project Object Model file called pom.xml

- Package a project
  - Change directory to root of project
  - mvn package
  - This generates a folder called target containing a snapshot.jar

Java Eclipse Project for AEM

- Maven Archetype for AEM
  - https://github.com/adobe/aem-project-archetype
- In Eclipse
  - File – New – Project – select AEM then AEM Sample Multi-Module project
  - Fill in details and generate project (takes ages)
  - The only details you must provide are
    - Name
    - Group Id
    - Artifact Id

Java Project Structure

- There are files that are needed for operational code and there are files that contain some test code and other resources
- Projects usually have two directories
  - src/main
  - src/test
- Java files are in src/main/java and src/test/java
- Resource files are in src/main/resources and src/test/resources

Java Packages

- A Package is a group of related classes and interfaces
- A package also corresponds to a physical directory on your computer
- When you compile a Java class, the .class file must be located in a directory named after the package name
- Package names can include dots and represent a hierarchical directory structure
- E.g. a package name of com.ctm.training would correspond to a directory structure of com/ctm/training

Importing Classes

- To use a class defined in a different package you have two options:
  - Use the fully-qualified class name everywhere
    - anotherPackage.AnotherClass obj = new anotherPackage.AnotherClass();
  - Import classes at the top of your code
    - package mypackage;
    - import anotherPackage.AnotherClass1; // direct access to AnotherClass1
    - import anotherPackage.AnotherClass2; // direct access to AnotherClass2
    - import anotherPackage.AnotherClass3; // direct access to AnotherClass3
    - import somePackage.*; // direct access to any class in somePackage

Java Packages and Access Modifiers

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Java Project Lifecycles

- Compile, Test, Package, and Deploy
- Usually abstracted away to the IDE or to build tools such as Maven

Java Object Oriented Programming

- Code is structured to encapsulate an idea, abstracted for convenience and inherited where possible
- Encapsulation
  - Every object exposes an API of public properties and methods but may contain its own private ideas of how to get stuff done
- Inheritance
  - As much as possible, shared ideas get abstracted to higher level objects which are inherited into more specialized objects as required
- Polymorphism
  - Where possible an interface can solve more than one related problem. The usage determines how

Java Compiler and Java Execution

- This is called javac and it takes human-readable .java files and compiles it to a .class file
- Then you can execute the .class file using a Java Virtual Machine

Java and JavaScript

- Java is very case sensitive
- Java must use data typing and return types
- Java is not as forgiving as JavaScript
- Statements must terminate with a semi colon
- Java is multi-threaded

Java is Object Oriented

- All code must be defined in a class
- File names are very important in Java
- Each public class must be in a .java file with the same name
  - HelloWorld.java

```
public class HelloWorld {

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

A Basic Java Program

- Create a file
- Write some Java
- Use javac to compile it
  - javac Example.java
- Use java to execute it (i.e. use a JVM called java)
  - java Example

Java Separators

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference.
...	Ellipsis	Indicates a variable-arity parameter.
@	Ampersand	Begins an annotation.

Java Keywords

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	exports	extends
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
module	native	new	open	opens	package
private	protected	provides	public	requires	return
short	static	strictfp	super	switch	synchronized
this	throw	throws	to	transient	transitive
try	uses	void	volatile	while	with
_					

Java Standard Classes

- Java has an extensive library of standard classes e.g.
  - System
  - Collections
  - File I/O
- Java EE (used by AEM) additionally supports
  - Web applications
  - Middle-tier components for transactions and security
  - Web services
  - Object-relational mapping
  - OSGi bundles

Standard Java Classes are Grouped by Functionality

- java.lang Essential classes (e.g. String), auto-imported
- java.text Text-processing classes (e.g. NumberFormat)
- java.io Input/output classes, streams, file-related classes
- java.sql Database-related classes
- java.util Utility classes (e.g. Date, Scanner, collections)
- javax.swing Swing UI classes (for Windows-based apps)

Java Primitive Data Types

- There are several primitives in Java
  - Integers
  - Floating Point numbers (float and double)
  - Characters
  - Booleans
- Each can represent single values, not complex objects
- All primitive types are passed-by-value into functions
- All other types are classes or interfaces and are passed-by-reference into functions

Java Has 8 Primitive Types

- All types have a defined range so they remain portable across platforms
  - byte: 1-byte whole number -128 to 127
  - short: 2-byte whole number -32,768 to 32,767
  - int: 4-byte whole number -2,147,483,648 to 2,147,483,647
  - long: 8-byte whole number
    -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
  - float: 4-byte floating-point number -3.4E38 to 3.4E38 7 sig figs
  - double: 8-byte floating-point number -1.7E308 to 1.7E308 16 sig figs
  - char: 2-byte (i.e. Unicode) character
  - boolean: must be true/false (can't use 1/0)

Operators

- The usual precedence applies, including braces

Flow Control

- If, elseif and else
- Switch case
- For loop
- While loop

Permitted Data Types

- A variable or function may be typed but then use a lower type
  - char ch1 = 88 // integer gets cast as a character (via its unicode value)

Scope

- By default variable are only in scope within their current block and blocks nested inside the current block
- Variables are only valid after they have been declared
- A variables will not hold a value once it is out of scope

Type Casting

- Types can be cast to a related type
- Automatic type conversion will take place if
  - The two types are compatible
  - The destination type is larger than the source type
- Automatic type casting
  - byte a = 20;
  - byte b = 40;
  - byte c = 100;
  - int d = a * b / c;
- However this can lead to problems
  - b = b * 2 // error cannot assign int to byte
- So use explicit type casting
  - b = (byte)(b * 2)

Type Promotion

- Java defines several type promotion rules that apply to expressions They are as follows
  - First, all byte, short, and char values are promoted to int
  - Then, if one operand is a long, the whole expression is promoted to long
  - If one operand is a float, the entire expression is promoted to float
  - If any of the operands are double, the result is double

String Objects are Immutable

- Once created, you can't change a String object
- All the String methods return a new String object
- Use StringBuilder or StringBuffer to build strings
- String literals ("like this") are held in a string constant pool
- When the compiler encounters a string literal in your code, it tries to resolve it in the string constant pool

Some Useful Public Methods in String

- int length() // Note this is a method
- char charAt(int index)
- boolean equals(String s2)
- boolean equalsIgnoreCase(String s2)
- String toString()
- String toLowerCase()
- String toUpperCase()
- String trim()
- String concat(String s2) // Also + and +=
- String substring(int begin)
- String substring(int begin, int end)
- String replace(char oldChar, char newChar)

The Math Class

- Contains many mathematical methods and static constants
- Some of the methods
  - sin(), cos(), tan(), sinh(), cosh(), tanh()
  - asin(), acos(), atan()
  - log(), log10(), exp()
  - max(), min(), abs()
  - ceil(), floor(), round()
  - pow(), sqrt(), random()
  - toDegrees(), toRadians()
- Some of the constants
  - PI, E

Defining a Class

- There can only be one public class per file
- Filename must be classname.java
- You can also define any number of non-public classes

Access Modifiers

- public
  - Accessible by anyone
  - Methods and constants are often public
- private
  - Accessible only by class itself
  - Data and helper methods are usually private
- protected
  - Accessible by class itself, subclasses, and classes in same package
  - Allow access to members that are hidden from general client code
- (No access modifier)
  - Accessible by class itself, and classes in same package

Defining Instance Variables

- A class can define any number of instance variables
  - Each instance will have its own set of these variables
- General syntax
  - Optional access modifier (default is package-level visibility)
  - Optional initial value

```
[public | private | protected] type variableName [= expression];
```

Defining Accessors and Mutators (getters and setters)

- Common practice is to define public getters and setters for private instance variables
- Allows external code to get and set values as properties
- Getter without setter provides read only access

The this Keyword

- The this keyword is a reference to the current object
  - Allows instance methods to explicitly access instance variables and instance methods on the current object
- Uses of this
  - Allows use of the same name for local vars and instance vars
  - Allows you to pass a "reference to self" into callback methods

```
public void setAccountHolder(String ah) {
  accountHolder = ah;
}
```

```
public void setAccountHolder(String accountHolder) {
  this.accountHolder = accountHolder;
}
```

Defining Instance Methods

- A class can define any number of instance methods
- They each operate on a particular instance

Overloading Methods

- A class can contain several methods with the same name, so long as the number (or types) or parameters is different
- This is called overloading

Overriding Methods

- A class can override a method defined in the base class
- This is an inheritance-related concept
- You must use the same method signature as in base class
- You should also annotate with @Override

Arrays

- Declare
  - data-type identifier[]
  - public static void main(String args[])
- Assign
  - data-type identifier[] = new data-type[size]
  - int primes[] = new int[n];
  - or
  - int[] primes = new int[n];

Garbage Collection

- Java handles deallocation automatically
  - this is called garbage collection
- When no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed
  - There is no need to explicitly destroy objects
- Garbage collection only occurs sporadically (if at all) during execution
  - It will not occur simply because one or more objects exist that are no longer used

Object-Oriented Programming

- Models a problem in terms of objects
  - Objects encapsulate functions and data
  - Helps engineer better software
  - Provides better facilities for data abstraction
  - Enforces high cohesion and low coupling
  - Provides better facilities for re-use
  - Models problem in its natural form
  - Makes systems more amenable to change
  - Software components (classes) have independent existence
  - Application may be largely disposable

Java Project Lifecycle

- Analysis
- Specification (e.g. UML)
- Write tests based on spec’
- Write code until tests pass
- Iterate and improve performance

Patterns and Anti Patterns

- Patterns recognize common best practices
- Anti patterns recognize common pitfalls

Class Path

- The Java class path is a list of directories to search when trying to locate classes
- The compiler uses the class path
  - to locate classes used in code
  - to verify classes are used correctly
- The JVM uses the class path
  - to load classes used by code
  - to execute the code defined in these classes

Java Naming Conventions

- Classes start with an uppercase letter, mixed case thereafter
  - E.g. BankAccount, Customer, String, BufferedReader
- Methods start with a lowercase letter, mixed case thereafter
  - E.g. toString(), getBalance(), calcInterest()
- Variables start with a lowercase letter, mixed case thereafter
  - E.g. balance, accountHolder, postCode
- final (i.e. constant) variables are uppercase
  - E.g. final double OVERDRAFT_LIMIT = 2000;

Exceptions

- Java exception handling is managed via five keywords
  - try
  - catch
  - throw
  - throws
  - finally

Throwable

- All exception types are subclasses of the built-in class Throwable
- Throwable overrides the toString( ) method defined by Object so that it returns a string containing a description of the exception

Multiple catch Clauses

- You can specify multiple catch clauses
  - Each can catch a different type of exception
- When an exception is thrown each catch statement is inspected in order
  - The first one whose type matches that of the exception is executed
- After one catch statement executes the others are by-passed

Throwing Exceptions

- It is possible to throw an exception explicitly using the throw statement
  - The flow of execution stops immediately after the throw statement

Using ‘throws’

- If a method is capable of causing an exception that it does not handle it must specify as much
  - Callers of the method can guard themselves against that exception
- Include a throws clause in the method declaration
  - A throws clause lists the types of exceptions that a method might throw

Using ‘finally’

- Create a finally block of code to be executed after a try/catch block has completed and before the code following the try/catch block
  - The finally block will execute whether or not an exception is thrown