Contents

NM HSD Java Upgrade Training Syllabus

Overview

This course is delivered as a series of instruction modules, intended to cover key enhancements and additions to the Java language and standard library—primarily focused on those introduced in Java versions 6 through 11, but also including practice/reinforcement work on concepts and techniques introduced in Java 5 and earlier versions.

Format

Each module consists of one or more primary subject units. Each unit will be covered in one or more of these formats:

- Independent reading of content prepared and published online by DDC, accompanied by an autograded programming assignment. (In some cases, instructor review will be required in addition to the autograding.)
- Real-time lectures with guided practical exercises, delivered in a video classroom.

Practical assessment will be administered approximately weekly, to confirm participants' working understanding of the concepts and techniques. These assessments will typically be introduced during real-time instruction, but will be open for code submissions over a 48-hour period. Code submitted will be read, compiled, tested, and graded by the instructors.

Schedule

The breakdown of modules by week is approximate. For example, some modules will begin at the end of one week and complete in the next; other modules will have some portions of the content introduced weeks before the module starts. Finally, the schedule is subject to adjustment as the course progresses.

Week start	Module
2/15	Basic Java skills refresher/reinforcement
2/22	Exception handing, Type inference, Module system
3/1	Test-driven development and unit testing refresher/reinforcement
3/8	Implementing methods in interfaces, Functional interfaces and lambdas
3/15	Streams framework
3/22	(none)
3/29	Spring/Spring Boot frameworks
4/5	Basic RDBMS access with JDBC
4/12	SQL Persistence with Spring Boot framework
4/19	Angular (w/ HTML, CSS, TypeScript/JavaScript) as REST service client
4/26	Version control and issue tracking
5/3	Continuous integration

Modules

Basic Java skills refresher/reinforcement

Overview

While this module is intended to serve as a broad review of a number of Java concepts and techniques, it focuses primarily on features introduced in Java versions 5 through 7, as well as other features that are particularly relevant to address in preparation for moving to Java 8 and beyond.

Units

- Classes (fields, methods, constructors/initializers, class & member modifiers); `static` nested & inner classes; local classes; anonymous classes.

- Using `import` and `import static` for accessing classes, interfaces, and static members from other packages.
- Pre-Java 8 interfaces (constants, method declarations).
- Inheritance and polymorphism.
- Annotation concepts & applications.
- `enum` (enumeration) classes.
- Generic classes, interfaces, and methods.
- Using the collections framework.

Learning outcomes

- Write interfaces, abstract classes, and concrete classes suitable for subsequent implementation/extension.
- Implement interfaces and extend classes, inheriting and overriding methods.
- Demonstrate understanding of generic classes, interfaces, and methods.
- Demonstrate understanding of conceptual and practical distinctions between *`Set`*, *`List`*, and *`Map`* interfaces, and between the most widely used implementations of those interfaces.

Exception handling

Overview

This module will review the exception-handling mechanisms of Java and the `Throwable` portion the Java class hierarchy, with particular focus on the post-Java 5 additions of multi-`catch` and `try`-with-resources.

Units

- Review of `try-catch-finally`, in concept and practice.
- Multi-catch (specification of multiple `Throwable` subclasses in `catch`).
- `try`-*with-resources*; `Closeable` and `AutoCloseable` interfaces.

Learning outcomes

- Demonstrate understanding (and conditions for use) of multi-catch syntax in `try-catch`.

- Demonstrate understanding of `try`-*with-resources* as an alternative to (or an augmentation of) `try-finally`, for automatic resource management.
- Implement the `Closeable` or `Autocloseable` interface for use in `try`-*with-resources*.

Type inference

Overview

This module deals with a small but important set of enhancements introduced in Java 7 & 10, which allow the compiler to infer type information in certain contexts.

Units

- Type inference in generics.
- Type inference in local variables.

Learning outcomes

- Use type inference in instantiation of generic classes.
- Demonstrate understanding of syntax and limitations of local variable type inference.

Module system

Overview

Many Java applications and libraries (including the Spring framework) rely on the bytecode inspection capabilities provided by the `java.lang.reflect` package. However, unrestricted inspection permits code to bypass access level declarations that would normally prevent outside access to class members. This module introduces Java *modules*, introduced in Java 9, which—among other benefits—permit an application to be organized into modules, each of which can be closed to outside inspection, or opened to inspection by a limited set of external modules.

Units

- Introduction to the Java 9+ module system.
- Controlling intra-JVM, inter-package, and reflection access with `module` declarations.

- Using the `jlink` tool to build a custom Java runtime library.

Learning outcomes

- Build a modular Java application incorporating access to and reflection by an external library.

Test-driven development and unit testing refresher/reinforcement

Overview

In this module, basic TDD and unit testing concepts will be reviewed, as well as specific techniques for creating unit tests with JUnit5, interactive use of JUnit5 in the build-debug-fix cycle, and build-integrated testing. (Continuous integration-based testing will be addressed in the "Continuous integration" module.)

Units

- Test-driven development and unit testing concepts.
- Writing and running JUnit5 tests.

Learning outcomes

- Demonstrate understanding of core TDD concepts and practices.
- Create and use unit tests as a key step in the development process.
- Demonstrate practical understanding of commonly used JUnit5 assertions and annotations.

Implementing methods in interfaces

Overview

This module introduces key Java 8 additions to the `interface` syntax. With that release, support for the definition of *concrete* (non-`abstract`) methods with the `default`, `static`, and `private` modifiers was added. This provides many benefits, including the ability to add methods to existing interfaces without breaking code implementing those interfaces; this capability is leveraged extensively in the streams framework (also introduced in Java 8).

Units

- `default`, `static`, `private` and `private static` methods.
- Multiple inheritance of behavior from interfaces; the *diamond problem.*

Learning outcomes

- Implement `static` and non-`static` methods in interfaces, with `private` methods as *helpers*.
- Disambiguate multiply-inherited `default` methods when implementing multiple interfaces in a `class`, or extending multiple interfaces in an `interface`.

Functional interfaces and lambdas

Overview

This module addresses arguably the most important Java 8 addition to the language: the *lambda* (sometimes called a *closure* or *anonymous function*). The contents of this module begin with anonymous classes (many of which may be replaced by lambdas) and *functional interfaces*, which provide the language infrastructure—as well as the constraints—for lambdas. The multiple syntactical options for lambdas are covered, as well as a number of applications from the standard library and in practical programming tasks.

Units

- Review of syntax and use of *anonymous classes* to extend classes or implement interfaces._
- Functional interfaces in concept and practice; key functional interfaces in standard library (*`Comparator`*, *`Function`*, *`Predicate`*, *`Consumer`*, *`Supplier`*, *`Runnable`*, etc.).
- Introduction of *lambdas* as implementations of functional interfaces; lambdas vs. anonymous classes.
- Alternative syntaxes: statement lambdas, expression lambdas, method references.
- Application of functional programming techniques to *`Collection`* and `Optional`.

Learning outcomes

- Sort collections using lambdas to enforce an ordering other than the elements' natural ordering.
- Remove items from a *`Collection`* with a lambda implementation of *`Predicate`*.
- Implement *`TaskTimer`* with a lambda to schedule a simple recurrent operation on a thread other than the main thread.
- Use lambdas in implementing unit tests for expected exceptions.
- Apply functional techniques to handling of `null` values using `Optional` with lambdas.

Streams framework

Overview

The streams framework, supporting functional-style operations on streams of elements (objects as well as a subset of primitive types) was added to the standard library in Java 8, building on the language enhancements for interfaces and lambdas. This module includes a variety of exercises, applying Java stream methods to perform map/filter/reduce and other data generation and processing operations.

Units

- *`Stream`* basics; introduction to lazy execution, stateless & stateful operations, intermediate & terminal operations.
- Creating streams from collections and arrays; generating streams with *`Supplier`*; collecting stream contents.
- Applications of the `filter(`*`Predicate`*`)` and `map(`*`Function`*`)` methods.
- Applications of the `flatMap(`*`Function`*`)` and `reduce(`*`BinaryOperator`*`)` methods.
- Primitive specializations of `Stream`,

Learning outcomes

- Demonstrate working understanding of lazy execution, as it relates to intermediate & terminal operations.
- Use `map`, `filter`, and `flatMap` to filter and transform data, including parsing data from input I/O streams.

- Use `reduce` or `collect` to categorize/group data and to construct and populate `Map`, `Set`, and `List` instances.
- Use a *`Supplier`* lambda to generate a stream of data values.

Spring/Spring Boot frameworks

Overview

This module introduces the Spring framework, including an overview of its implementation of key aspects of the *inversion of control* principle. Most of the practical work in this module uses the Spring Boot framework, which incorporates Spring, along with a number of best-of-breed libraries, and a powerful autoconfiguration capability; together, these provide a rapid onramp into development of enterprise-level services and applications

Units

- Introduction to inversion of control (IoC) and dependency injection; introduction to Spring stereotypes and autowiring.
- Creating projects using Maven & Maven archetypes, Spring Initializr; introduction to Spring Boot starter libraries; core Spring Boot plugins for Maven.
- Developing a simple REST service with Spring MVC.
- Using templates to provide web page-based access to a Spring MVC application.

Learning outcomes

- Demonstrate basic understanding of Spring & Spring Boot concepts by creating a simple Spring Boot-based application with REST and web page endpoints.

Basic RDBMS access with JDBC

Overview

JDBC is an API for platform-independent access by Java applications to relational databases. This module presents and exercises the fundamental concepts, classes, and interfaces of JDBC, while also preparing the student to working with libraries that build on JDBC to provide higher-level capabilities.

Units

- Review of RDBMS & SQL concepts; introduction to JDBC drivers and connections.
- – Using *`PreparedStatement`* and *`ResultSet`*.
- Mapping SQL structured types to *`Struct`* and to domain classes with custom type maps.

Learning outcomes

- Implement a simple relational data model w/ JDBC, using *`PreparedStatement`* and *`ResultSet`* for type- and injection-safe execution.

SQL Persistence with Spring Boot framework

Overview

This module builds directly on the previous, starting with the use of Spring Data JDBC to incorporate relational persistence into Spring Boot-based applications. The *Java Persistence API*, an object-relational mapping (ORM) specification, is then introduced through exercises with Spring Data JPA and the Hibernate implementation of JPA.

Units

- Introduction to Java Persistence API (JPA) & Hibernate implementation.
- Using Spring Data JDBC for ORM & CRUD operations.
- Using the Hibernate implementation of the Java Persistence API (JPA) for ORM w/ Spring Data JPA for CRUD operations & inferred queries.
- Implementing a persistent REST web service w/ Spring Boot, Spring MVC, & Spring Data.

Learning outcomes

- Implement a simple relational data model w/ JPA, Hibernate, and Spring Data.
- Expose a relational data model as a REST web service with Spring Boot MVC.

Angular (w/ HTML, CSS, TypeScript/JavaScript) as REST service client

Overview

This module includes the development and refinement of a simple Angular CLI-based web application, as a client to the web service developed in the previous modules.

Units

- Overview of HTML & CSS concepts & techniques.
- Overview of JavaScript & TypeScript.
- Introduction to Angular & Angular CLI.
- Implementation of Angular controller for consuming REST web service.
- UI customization with CSS & Bootstrap.

Learning outcomes

- Develop & customize (using CSS) simple HTML pages.
- Demonstrate basic practical understanding of REST client development using Angular, TypeScript/JavaScript, and Bootstrap, through development & customization of a client application for Spring-based REST service

Version control and issue tracking

Overview

While version control systems (VCS) are a key component of the technical infrastructure for this entire course, this module takes version control (particularly Git) and VCS-integrated issue tracking (using Jira), as central topics. (A number of concepts and techniques included in this module—such as basic practices for using VCS—will first be introduced earlier in the course.)

Units

- Review of basic version control concepts and Git.
- Practical branching and merging; resolving merge conflicts.
- Forking and pull requests.
- Introduction to issue tracking with Jira.

- Integrating issue tracking with version control.

Learning outcomes

- Demonstrate practical understanding of core Git operations: creating & cloning repositories; committing changes; pushing to & pulling from remotes; merging & resolving merge conflicts.
- Use BitBucket for remote repository management, including forking & merging, with & without pull requests.
- Use version control operations with Jira issue tracking.

Continuous integration

Overview

This module builds on the version control and TDD modules to introduce concepts and practices in continuous integration (CI). Hands-on work will include basic workflow automation for integrating automated testing and deployment into the development process.

Units

- Introduction to CI with Jenkins.
- Configuring build & test tasks with Jenkins.
- Integrating Jenkins & Jira.
- Automatic deployment of build artifacts to Maven repository.

Learning outcomes

- Demonstrate understanding of CI concepts.
- Create & modify Jenkins configurations for automated unit testing of Java projects.
- Connect reporting & workflow automation in Jira and Jenkins.