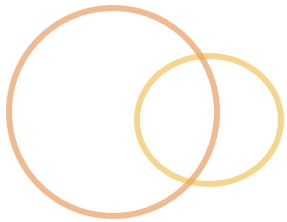
What's New In Java

An Overview of Tool Enhancements

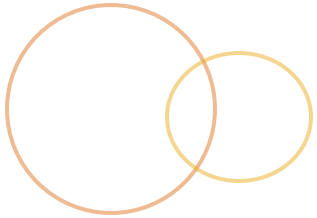
Presentation Topics

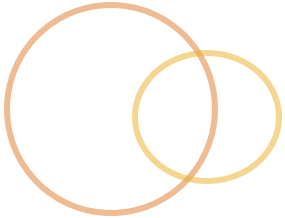


In this section, we will:

- Explore enhancements made to the Java compiler
- Discuss enhancements made to Javadoc
- Introduce the Annotation Processing Tool (apt)

Objectives

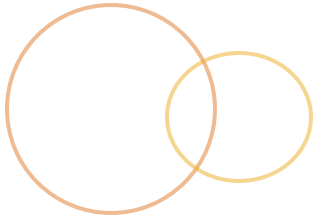


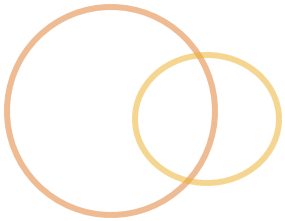


When we are done, you should be able to:

- List 2 enhancements made to the Java compiler
- Explain the purpose of APT

Java Compiler Enhancements





Java Compiler Enhancements

- Compiler enhanced to support new language features
  - Converts “foreach” into `for`
  - Converts static imports into fully-qualified references
  - Supports new `enum` and Annotation types
  - Auto-boxing aware

Java Compiler Enhancements [cont.]

- Supports new compile-time flags
  - `-Xlint`
    - lint - as in dryer dust
    - Enables compiler to produce warnings about potentially problematic code; like missing `finally` blocks
    - Possible usages:
      - `-Xlint / -Xlint:none` - enable / disable all warnings
      - `-Xlint:xxx / -Xlint:-xxx` - enable / disable specific warning
      - Valid warnings:
        `all,deprecation,unchecked,fallthrough,path, serial,finally`

Java Compiler Enhancements [cont.]

- Supports new compile-time flags
  - `-source version`
    - Provides source compatibility with version
  - `-target version`
    - Provides class compatibility with version
- Supports cross-compilation - change `classes.zip` used to compile source

Lab

- **Description**: Write an application (`LintTest`) that tests the “lint” detector in the compiler
  - [a] Test the lint detection on switches (fall through)
  - [b] Test the lint on a finally block (missing)
  - Turn off lint detection when you compile
- **Duration**: 20 minutes

LintTest Solution [a]

```
1   package labs.solutions.toolenhancements;
2
3   /**...*/
13  public class LintTest {
14      public static void main(String[] args) {
15
16          if(args.length == 0) {
17              System.out.println("LintTest can not run; please specify a command line argument.");
18              System.exit(0);
19          }
20
21          int x = Integer.parseInt(args[0]);
22          switch(x) {
23              case 0:
24                  System.out.println("Case 0 was called");
25              case 1:
26                  System.out.println("Case "+ x + " was called");
27              case 2:
28                  System.out.println("Case "+ x + " was called");
29                  break;
30              default:
31                  System.out.println("Default Case was called");
32                  break;
33          }
34      }
35  }
36
```

```
[345] >javac -Xlint LintTest.java
LintTest.java:25: warning: [fallthrough] possible fall-through into case
            case 1:
            ^
LintTest.java:27: warning: [fallthrough] possible fall-through into case
            case 2:
            ^
2 warnings
[346] >
```

Java Compiler Enhancements [cont.]

- Sun's Compiler now supports programmatic invocation
  - Don't need to invoke `Runtime.exec` to invoke compiler
  - Can invoke compiler through method call
    - `com.sun.tools.javac.Main.compile`
  - Successful compilation generates `.class` file
- Useful if program generates source code and needs to compile it

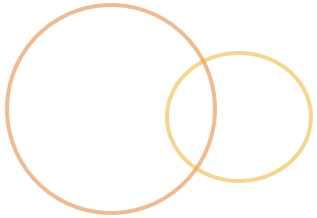
Java Compiler Example

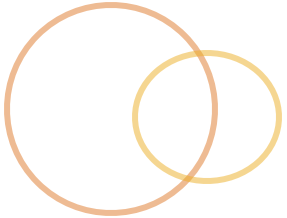
```
1  package examples.compiler;
2
3  import java.io.PrintWriter;
4  import java.io.StringWriter;
5
6  /**
7   * The following example
8   * illustrates invoking the
9   * compiler using compiler API
10  * instead of Runtime.exec
11  */
12 public class Compiler {
13
14   public static void main(String[] args) {
15     StringWriter sw = new StringWriter();
16     PrintWriter pw = new PrintWriter(sw);
17     int compiled = com.sun.tools.javac.Main.compile(args, pw);
18     System.out.println("Compile output: " + compiled);
19     System.out.println(sw.getBuffer());
20   }
21 }
22
```

Javadoc

An Quick Overview

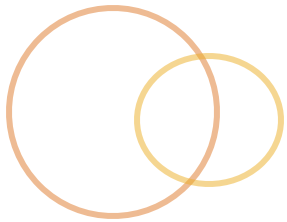
Javadoc





- What is Javadoc?
  - Documentation generation tool for Java
  - Generates well-defined documentation structure from source-code
  - Documentation contains information on:
    - Packages
    - Types
    - Members
    - Methods
  - Used to generate Java APIs

Javadoc [cont.]



- Why is it needed?
  - Need a formal way to convert source-code level documentation into user-level documentation
  - Frees documentation from source-code
  - Code is not self-documenting!

Javadoc [cont.]

- How does it work?
  - Stand-alone Java program
  - Relies on Doclet framework
    - Doclets are Java programs
    - Use doclet API
    - Specify content and format of the output for Javadoc
    - Sun provides “standard” doclet
  - Uses `javac` to process source files
  - Relies on specific documentation syntax and annotations found in source code

Javadoc Documentation Syntax

- Uses special style of comment
  - Known as a doc comment; multi-line c-like style comment

```
/**
*
*/
```

  - Doc comment precedes type, member, and method declarations
  - Supports “annotations” to identify specific things

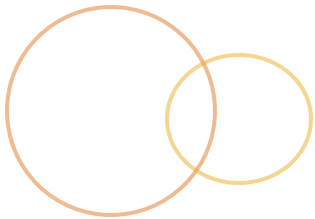
Doc Comment Structure

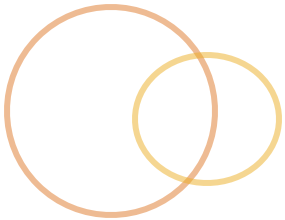
- Well-defined structure within doc comment block
  - **First entry** - one sentence summary
  - **Second entry** - additional description (may include HTML); considered descriptive paragraphs
  - **Third entry** - javadoc “annotations”; each on single line
    - `@author` - author of code
    - `@version` - version of code
    - `@since` - when code was introduced
    - `@param` - description of parameter
    - `@return` - description of return value
    - `@throws` - description of throws clause

Javadoc Enhancements

- Javadoc enhanced to support new language features
  - Supports Generics, Enums, and varargs
  - Supports Annotations
- Introduces new tags
  - `@literal`
  - `@code`
- Introduces `package-info.java`
  - Used to annotate package statement
  - Also contains package information for `package.html`

Annotation Processing Tool





Annotation Processing Tool

- Command-line utility (`apt`) for processing annotations
- Provides set of APIs required to process annotations
- Starting point for creating own annotation processing tool
  - Define own annotations
  - Create class that implements `AnnotationProcessorFactory`
  - Tell `apt` which annotations to process; looks for appropriate processor factory

Summary

Development tools enhancements

- Development tools modified to support new language features
- `javac` modified to support “lint”
- `apt` introduce to make annotation processing extensible