Fast Track to Java

Customized for Starbucks

Delivered by DevelopIntelligence

Java Packages

Objectives

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

- Understand what Java packages do
- Interpret and using fully qualified class names
- Use the ***`package`*** statement correctly
- Use the ***`import`*** statement correctly
- Understand how ***`import`*** on demand works
- Lay out directories for packages
- Use classpath to locate class binaries
- Understand and use static imports

Packages

- Allow developers to encapsulate collections of related classes and interfaces into larger aggregations
- Do not exist as objects or concrete constructs in the way that classes or interfaces exist
- Exist as logical groupings of classes
- Described in a way understood by JVM – namespace

Packages (cont.)

- Java SE, Java EE, Java ME are collections of packages

- Java SE provides the core packages for the language
  - ***java.lang***
  - ***java.net***
  - ***java.util***

- Java EE and Java ME provide packages that are extensions to the language
  - ***javax.ejb***
  - ***javax.servlet***
  - ***javax.message***

Java Packages Perspectives

Two perspectives to consider when thinking about packages

- **Design**
  - How to choose packages
  - How to choose classes for packages
  - How to choose package interfaces
- **Implementation**
  - How packages are defined
  - How packaged classes are accessed in code
  - How the compiler and JVM manage and work with packages

Java Package Design

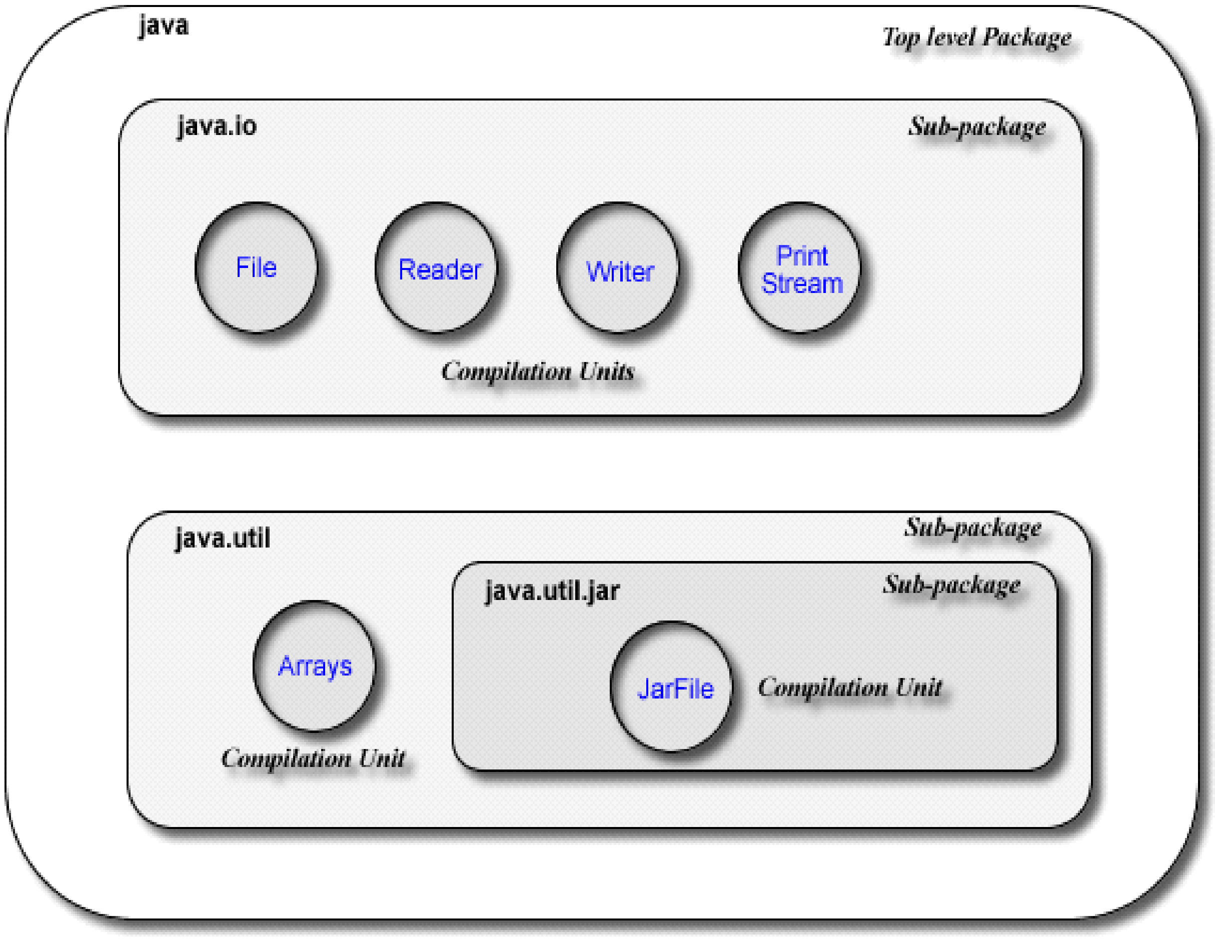
- Package names should provide some human-understandable grouping of classes
  - Can have multiple levels separated by periods
  - Each level must be a valid Java identifier
  - Convention uses only ASCII lower case letters
- Packages are part of namespace system
  - Used by the class loading and security mechanisms
  - Namespaces qualify classes,
    - E.g. ***`java.sql.Date`*** and ***`java.util.Date`***
- ***`java`*** and ***`javax`*** prefixes are reserved

Java Package Design (cont.)

- Reverse your domain name for your prefix
  - ***com.developintelligence.***
  - ***com.apple.***
  - ***com.level3.***
- Determine the sub-packages
  - Sub-packages are logical, not physical
  - Types of groupings
  - Order from most generic to most specific

```
com.developintelligence.training.java.intro.labs
com.developintelligence.training.java.intro.solutions
com.developintelligence.bankapp
com.developintelligence.bankapp.util
```

Defining Java Packages



Java package organization

Package Implementation

- Every class belongs to exactly one package
  - Explicit package statement
  - Implicit – becomes part of *default/unnamed package*
- Classes are tied to a package in their source
  - Include a package statement as first executable line in code
  - Can only be one package statement per source file

```
package com.developintelligence.sky;
class Blue {
  /* body */
}
```

- Package may contain unlimited classes
- *default* and `protected` access are about package membership

Accessing Classes in Packages

There are three scenarios when accessing classes

1. Accessing a class in the same package
   - Use short name of the class, e.g. Date, BankAccount
   - Have access to classes in the same package
2. Accessing class belonging to a different package than the class itself
   - Use *fully-qualified class name*
   - Use an ***`import`*** *statement*
3. Accessing class in ***`java.lang`***
   - Use short name, this namespace is always visible

Fully Qualified Class Name Example

```
public class FullyQualified {

  public static void main(String [] args) {
    java.util.Date d1 = new java.util.Date(8987811L);
    java.sql.Date d2 = new java.sql.Date(8987811L);
    System.out.println(“java.util.date is " + d1);
    System.out.println(“java.sql.date is " + d2);
  }
}

// Output of the above is
java.util.date is Wed Dec 31 21:29:47 EST 1969
java.sql.date is 1969-12-31
```

Importing Classes

- Using fully qualified class names works
  - Very explicit
  - Easy to read, maintain
  - Laborious to type (though IDE might help)
- Importing classes is a short cut
  - Use an ***`import`*** *statement*
    - ***`import`*** follows the package statement
  - Gives class access to classes in other packages
  - Might experience class name collisions
  - ***`import`*** classes or whole packages

```
import java.net.Socket;  //access to single class
import java.util.*;  //access to all classes
```

  - Compiled code uses only fully qualified class names

import Statement Example

```
import java.util.Date;
public class DateImporter {
  public static void main(String [] args)  {
    Date d1 = new Date(8987811L);
    java.sql.Date d2 = new java.sql.Date(8987811L);
    System.out.println("java.util.date is " + d1);
    System.out.println("java.sql.date is " + d2);
  }
}

// Output of the above is
java.util.date is Wed Dec 31 21:29:47 EST 1969
java.sql.date is 1969-12-31
```

The import On Demand

```
import java.util.*;
  public class WildImport {
    public static void main(String [] args)  {
      Date d1 = new Date(8987811L);
      java.sql.Date d2 = new java.sql.Date(8987811L);
      System.out.println("java.util.date is " + d1);
      System.out.println("java.sql.date is " + d2);
    }
  }
  // Output of the above is
  java.util.date is Wed Dec 31 21:29:47 EST 1969
  java.sql.date is 1969-12-31
```

- Note that ***`import java.sql.*`*** in this example would render ***`Date`*** unusable—all references would have to be *fully qualified*

Environment Constraints

- Packages map to directory structures
  - The source for classes defined in packages ***should*** exist in a directory structure mapped to the packages
  - Classes ***must*** be placed into a directory structure mapping to the package structure, though it may be on a different root

Environment Constraints

- Compiler and JVM use variable to find classes belonging to packages
  - Can be environment variable ***`CLASSPATH`***
  - Can be passed to compiler and JVM as arguments: ***`-cp`*** or ***`-classpath`***
- Classpath describes paths to roots of your packages
  - Often a path to directory structure
  - Can also be path to an *archive*, like a ***`JAR`*** or ***`ZIP`***
  - ***`CLASSPATH=/myjava/bin:/ourjava/bin:/support/onezip.zip:/support/morejava.jar`***
  - Searched in order, left to right, first match wins

Static Imports

- What are they?
  - Mechanism for importing static variables and methods
  - Very similar to standard import syntax
- Why do they exist?
  - Simplify access to static variables and methods in code
  - Restore cohesion problem found in “work-around” solutions
- Are they still relevant?
  - Designed to shorten typing, but IDE now does most of this
  - Breaks rules of how to read Java, so might not be overall benefit

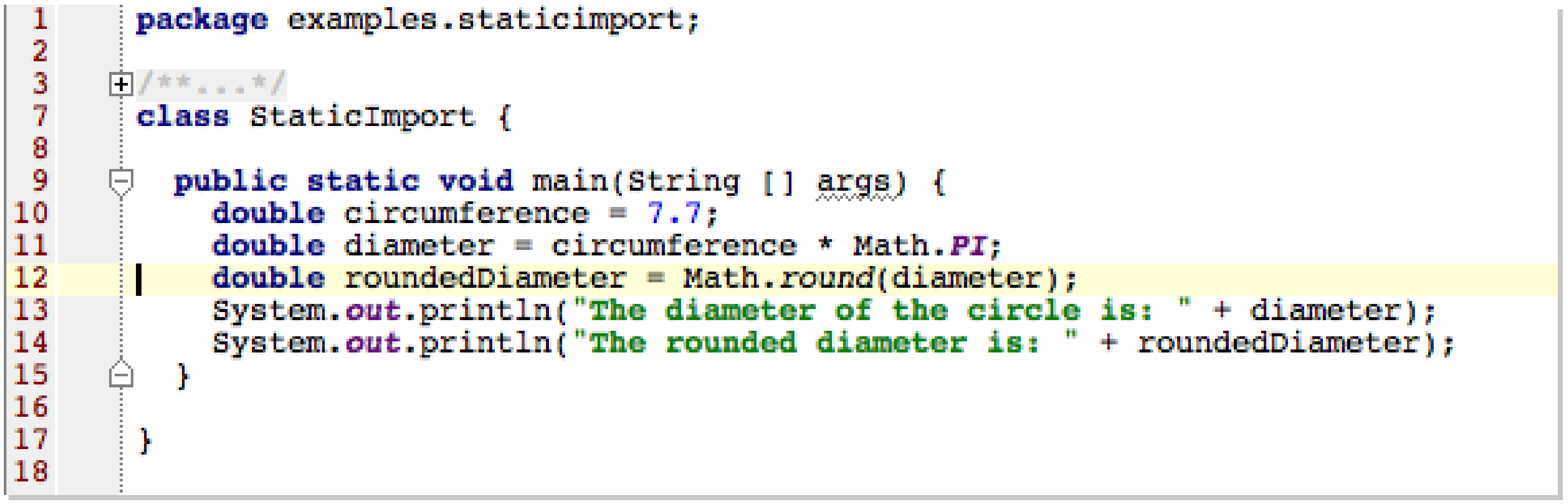
Static Imports [cont.]

- How do they work?
  - Like normal import mechanism
    - Development-time short-cut
    - Compiler converts short-cuts into fully qualified names
  - In static imports
    - Compiler converts “static” short-cuts into fully qualified names

Working With Static Imports

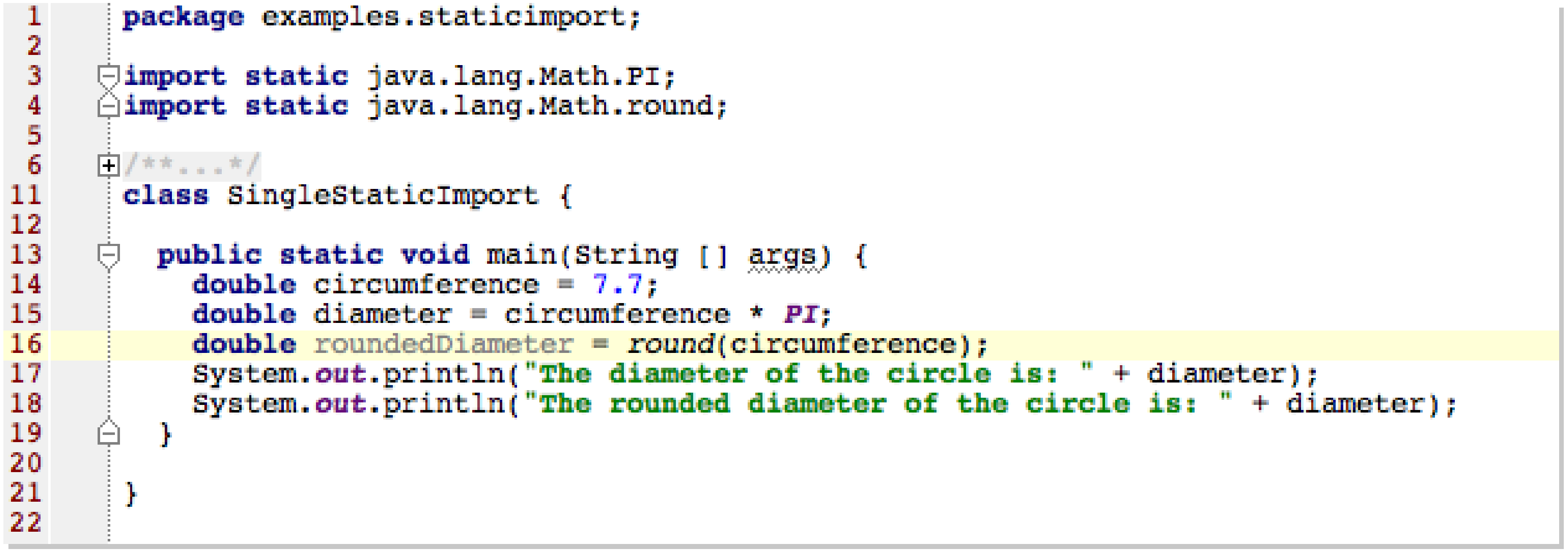
- Two types of static import
  - Single static import declaration
  - Static “on-demand” import declaration
- Look similar to . . .
  - Single type import declaration
  - “On-demand” type import declaration
- . . . but work a little different
  - Single static import - imports single static variable or function
  - “On-demand” static import - imports all static variables and functions

Static Import Example [old way]



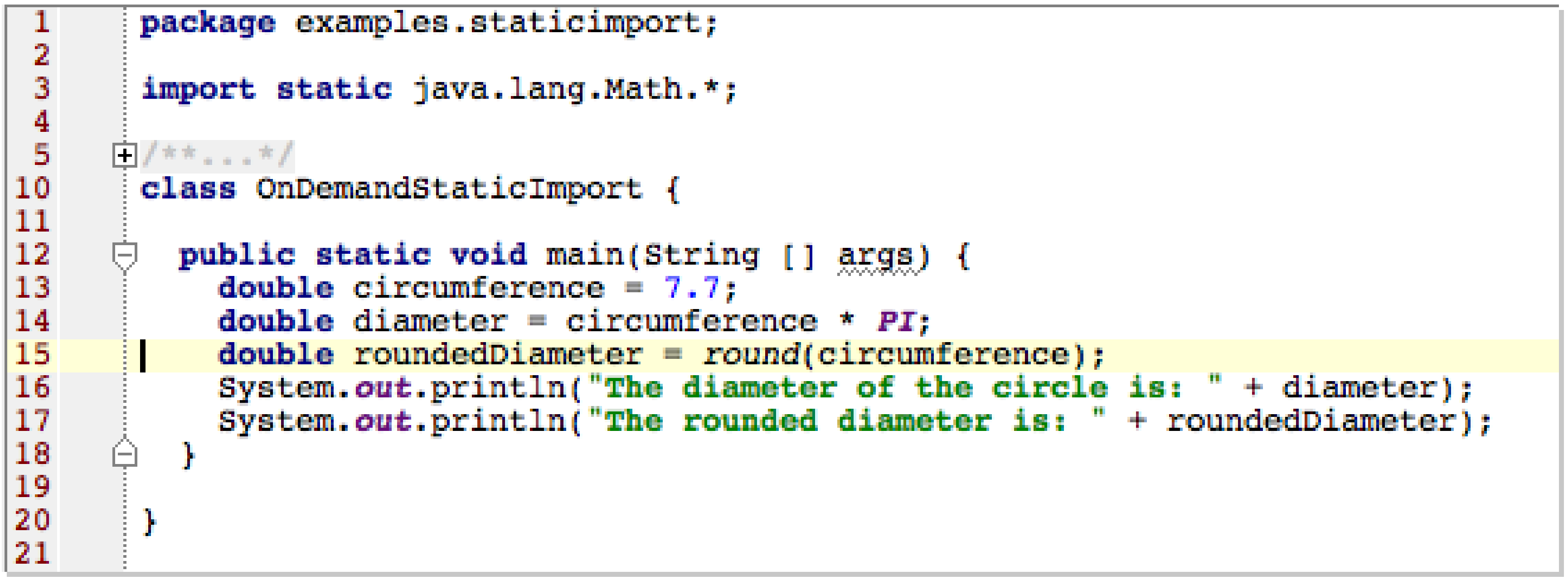
```
package examples.staticimport;

/**...*/
class StaticImport {

  public static void main(String [] args) {
    double circumference = 7.7;
    double diameter = circumference * Math.PI;
    double roundedDiameter = Math.round(diameter);
    System.out.println("The diameter of the circle is: " + diameter);
    System.out.println("The rounded diameter is: " + roundedDiameter);
  }

}

```

Single Static Import Example



```
package examples.staticimport;

import static java.lang.Math.PI;
import static java.lang.Math.round;

/**...*/
class SingleStaticImport {

   public static void main(String [] args) {
      double circumference = 7.7;
      double diameter = circumference * PI;
      double roundedDiameter = round(circumference);
      System.out.println("The diameter of the circle is: " + diameter);
      System.out.println("The rounded diameter of the circle is: " + diameter);
   }

}
```

On-Demand Static Import Example



```
package examples.staticimport;

import static java.lang.Math.*;

/**...*/
class OnDemandStaticImport {

  public static void main(String [] args) {
    double circumference = 7.7;
    double diameter = circumference * PI;
    double roundedDiameter = round(circumference);
    System.out.println("The diameter of the circle is: " + diameter);
    System.out.println("The rounded diameter is: " + roundedDiameter);
  }

}
```

Static Import Best-Practices

- Be aware
  - Name-space collisions can occur
  - Code can be hard to read
  - Breaks rules of how to read Java!
- Be specific
  - Consider avoiding wildcard notation
  - Use “optimize imports” functionality of IDE
- Avoid abuse
  - Perform proper OOAD anytime you create a static
- Refactor old code

Summary

We covered

- What Java packages do
- Interpreting and using fully qualified class names
- Using ***`package`*** statement correctly
- Using ***`import`*** statement correctly
- How ***`import`*** on demand works
- How to lay out directories for packages
- How to use classpath to locate class binaries
- Static imports

Lab 7

- Packages
  - Put your [*]Person classes and your application class into separate packages, e.g. *com.yourbiz.hr.domain.Person*, and *com.yourbiz.hr.app.MainApplication*.
  - Which means that you are first going to have to create a new Exception class called InvalidDateException.