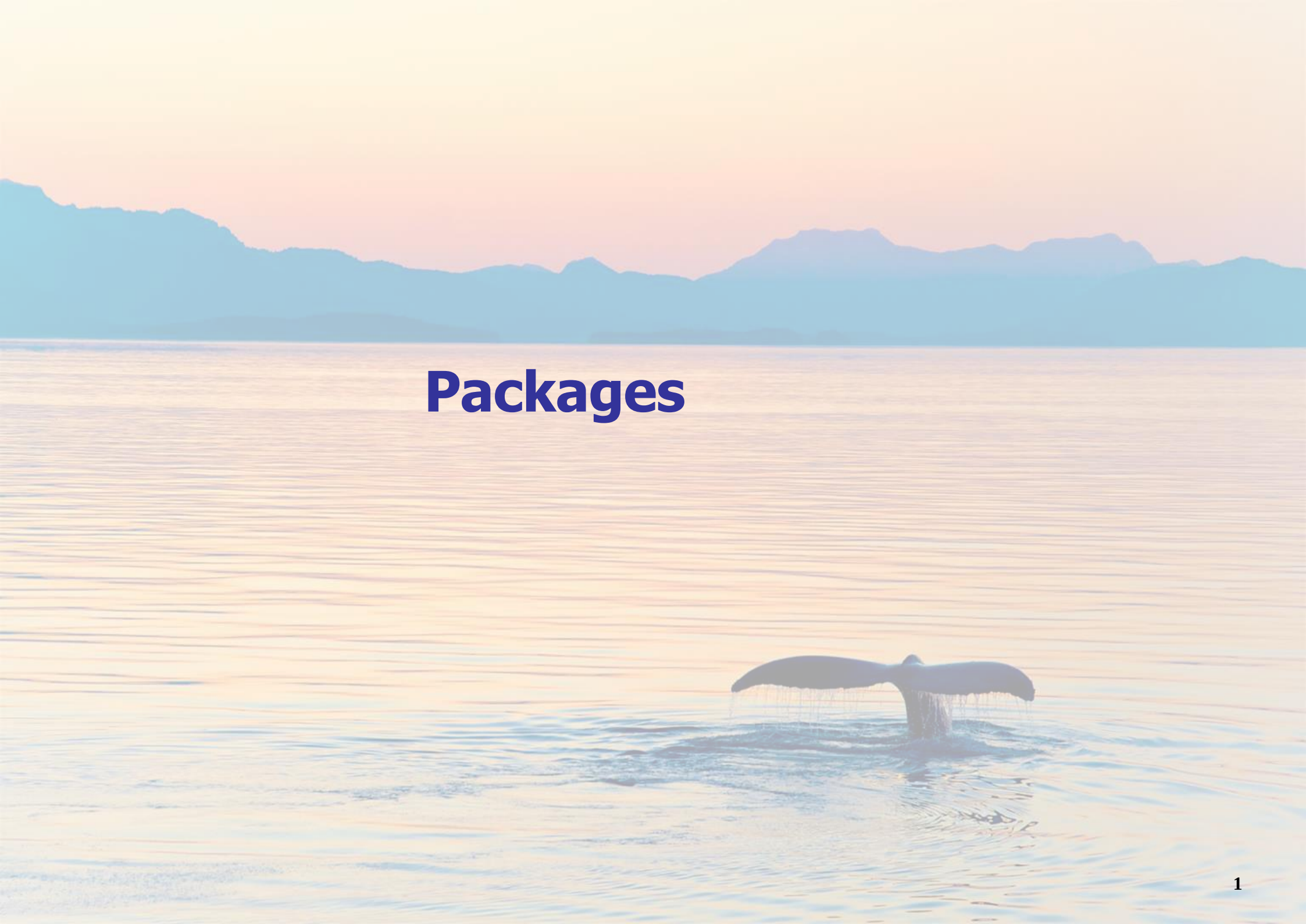

Packages

Contents

- What is a Package?
- Creating a Package
- Referring to a Package Member
- Managing Source and Class Files
- Summary of Creating and Using Packages

What is a Package?

- A package is a namespace that organizes a set of related classes and interfaces.
- Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another.

What is a Package?

- Because software written in Java can be composed of *hundreds* or *thousands* of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.
- The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications. This library is known as the "Application Programming Interface", or "API" for short.
- This allows the programmer to focus on the design of the particular application, rather than the infrastructure required to make it work.

What is a Package?

- The Java Platform API Specification contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java Platform 6, Standard Edition.
  - http://java.sun.com/javase/6/docs/api/
  - http://java.sun.com/javase/ja/6/docs/ja/api/
- It will become your single most important piece of reference documentation.
- **Definition:** A *package* is a grouping of related types providing access protection and name space management.

Creating a Package

- Suppose you write a group of classes that represent graphic objects, such as circles, rectangles,lines, and points.
- You also write an interface, Draggable, that classes implement if they can be dragged with the mouse.
- Your source code is in 6 files (see the right side of the slide)

```
//in the Draggable.java file
public interface Draggable {
   . . .
}
//in the Graphic.java file
public abstract class Graphic {
   . . .
}
//in the Circle.java file
public class Circle extends Graphic implements Draggable {
   . . .
}
//in the Rectangle.java file
public class Rectangle extends Graphic implements Draggable
{
   . . .
}
//in the Point.java file
public class Point extends Graphic implements Draggable {
   . . .
}
//in the Line.java file
public class Line extends Graphic implements Draggable {
   . . .
}
```

Creating a Package

- You should bundle these classes and the interface in a package for several reasons, including the following:
  - You and other programmers can easily determine that these types are related.
  - You and other programmers know where to find types that can provide graphics-related functions.
  - The names of your types won't conflict with the type names in other packages because the package creates a new namespace.
  - You can allow types within the package to have unrestricted access to one another yet still restrict access for types outside the package.

Creating a Package

- Put If you put the graphics interface and classes listed in the slide in a package called graphics, you would need six source files (see the right side of the slide)

- Put these files into the *graphics* folder and compile them:

  javac graphics/*.java

- **Note:** If you put multiple types in a single source file, only one can be public, and it must have the same name as the source file. For example, you can define public class Circle in the file Circle.java

```
package graphics;       //in the Draggable.java file
public interface Draggable {
   . . .
}
package graphics;       //in the Graphic.java file
public abstract class Graphic {
   . . .
}
package graphics;       //in the Circle.java file
public class Circle extends Graphic implements Draggable {
   . . .
}
package graphics;       //in the Rectangle.java file
public class Rectangle extends Graphic implements Draggable {
   . . .
}
package graphics;       //in the Point.java file
public class Point extends Graphic implements Draggable {
   . . .
}
package graphics;       //in the Line.java file
public class Line extends Graphic implements Draggable {
   . . .
}
```

Naming Conventions

- Package names are written in *all lowercase* to avoid conflict with the names of classes or interfaces. Companies use their reversed Internet domain name to begin their package names:
  - com.example.orion for a package named orion created by a programmer at example.com.
- In some cases, the internet domain name may not be a valid package name. The suggested convention is to add an underscore.

Legalizing Package Names

Domain Name	Package Name Prefix
clipart-open.org	org.clipart_open
free.fonts.int	int_.fonts.free
poetry.7days.com	com._7days.poetry

Referring to a Package Member

- To use a public package member (class, interface) from outside its package, you must do one of the following:
  - Import the package member.
  - Import the member's entire package.
  - Refer to the member by its fully qualified name.

Referring to a Package Member by Its Simple Name

- You can *import* the member of the package or the whole package and then use the simple name of that member
- The asterisk in the import statement can be used only to specify all the classes within a package, as shown here. It cannot be used to match a subset of the classes in a package

```
// importing the member
import graphics.Rectangle;
. . .
Rectangle myRectangle = new Rectangle();
…
```

```
// importing the whole package
import graphics.*;
. . .
Circle myCircle = new Circle();
Rectangle myRectangle = new Rectangle();
…
```

```
import graphics.A*; //does not work; error
```

Packages Imported Automatically

- The Java compiler automatically imports three entire packages for each source file (you do not need to specify the import statement):
  - the package with no name,
  - the java.lang package, and
  - the current package (the package for the current file).
- Members of these packages can be referred by their simple names.

Referring to a Package Member by Its Qualified Name

- If you do not use the import statement, you have to refer to the package member by its fully qualified name (see Slide 11, example 1).

```
graphics.Rectangle myRect = new graphics.Rectangle();
```

Apparent Hierarchies of Packages

- At first, packages appear to be hierarchical, but they are not. For example, the Java API includes:
  - a java.awt package,
  - a java.awt.color package,
  - a java.awt.font package.
- The prefix java.awt (the Java Abstract Window Toolkit) is used for a number of related packages to make the relationship evident, but not to show inclusion.
- The following statement imports all of the types in the java.awt package, but it *does not import* java.awt.color, java.awt.font:
  - import java.awt.*;
- To use the members of the aforementioned packages, you have to write:

```
import java.awt.*;
import java.awt.color.*;
Import java.awt.font.*;
```

Managing Source and Class Files

- Many implementations of the Java platform rely on hierarchical file systems to manage source and class files.
- The strategy is as follows:
  - Put the source code for a class or interfacein a text file whose extension is .java

```
// in the Rectangle.java file
package graphics;
public class Rectangle() {
    . . .
}
```

  - Then, put the source file in a directory whose name reflects the name of the package to which the type belongs:

```
...../graphics/Rectangle.java
```

Simple Example

```
// file ClassOne.java in the directory
//  /home/s111111/java/Ex08/demopackage
package demopackage;
public class  ClassOne {
    public void methodClassOne() {
        System.out.println("methodClassOne");
    }
}

// file ClassTwo.java in the directory
//  /home/s111111/java/Ex08/demopackage
package demopackage;
public class  ClassTwo {
    public void methodClassTwo() {
        System.out.println("methodClassTwo");
    }
}
```

- **Compilation:**
  javac *.java

```
// file UsageDemoPackage.java in
// the directory
//  /home/s111111/java/Ex08/
import demopackage.*;
class UsageDemoPackage {
    public static void main(String[] args) {
        ClassOne v1 = new ClassOne();
        ClassTwo v2 = new ClassTwo();
        v1.methodClassOne();
        v2.methodClassTwo();
    }
}
```

- **Compilation:**
  javac UsageDemoPackage.java
- **Run:**
  java UsageDemoPackage

Managing Source and Class Files

- The qualified name of the package member and the path name to the file are parallel:
  - class name: graphics.Rectangle
  - pathname to file: graphics¥Rectangle.java
- If the Example company had a com.example.graphics package that contained a Rectangle.java source file, it would be contained in a series of subdirectories like this:
  - …./com/example/graphics/Rectangle.java

Managing Source and Class Files

- When you compile a source file, the compiler creates a different output file (its extension is .class) for each type defined in it

```
// in the Rectangle.java file
package com.example.graphics;
public class Rectangle{
    . . .
}
class Helper{
    . . .
}
```

- the compiled files will be located at:

<path to the parent directory of the output files>/com/example/graphics/Rectangle.class
<path to the parent directory of the output files>/com/example/graphics/Helper.class

Managing Source and Class Files

- The full path to the classes directory is called the *class path*
- You may set the class path with the CLASSPATH system variable

The class path is:
<path_two>/classes

Package name is:
com/example/graphics,

The compiler and JVM look for .class files in
<path_two>/classes/com/example/graphics

- By default, the compiler and the JVM search the current directory and the JAR file containing the Java platform classes so that these directories are automatically in your class path.

Summary of Creating and Using Packages

- To create a package for a type, put a package statement as the first statement in the source file that contains the type (class or interface).
- To use a public type that is in a different package, you have three choices:
  - use the fully qualified name of the type,
  - import the type, or
  - import the entire package of which the type is a member.
- The path names for a package's source and class files mirror the name of the package.
- You might have to set your CLASSPATH so that the compiler and the JVM can find the .class files for your types.