CENG 1004
Introduction to Object Oriented Programming

Spring 2017

WEEK 6

Packages

Packages

- Each class belongs to package
- Classes in the same package serve a similar purpose
- Packages are just directories
- Classes in other packages need to be imported
- Classes that are not explicitly put into a package are in the “default” package.

Packages

Defining Packages

```
package path.to.package.foo;
class Foo {
    ...
}
```

Using Packages

```
import path.to.package.foo.Foo;
import path.to.package.foo.*;
```

Packages

- If the class is in a package named **test.pkg**,
  - then the first line of the source code will be

    package test.pkg;
  - the source code file must be in a subdirectory named “pkg” inside a directory named “test”

    /test/pgk/ClassName.java
  - Use “javac test/pkg/ClassName.java” to compile
  - Use “java test.pkg.ClassName” to execute

Why Packages?

- Group similar functionality
  - org.boston.libraries.Library
  - org.boston.libraries.Book
- Seperate similar names
  - shopping.List
  - packaging.List

Special Packages

- All classes see classes in the same package (No need to import)

- All classes see classes in java.lang
  - Example: java.lang.String; java.lang.System

Java API

Java API

- Java includes lots of packages/classes
- Reuse classes to avoid extra work
- http://docs.oracle.com/javase/8/docs/api/

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

What if the library expands?

ArrayList

Modifiable list

Internally implemented with arrays

Features

- Get/put items by index
- Add items
- Delete items
- Loop over all items

Array → ArrayList

```
Book[] books =
      new Book[10];
int nextIndex = 0;


books[nextIndex] = b;
nextIndex += 1;
```

```
ArrayList<Book> books
= new ArrayList<Book>();


books.add(b);
```

```
import java.util.ArrayList;
class ArrayListExample {
   public static void main(String[] arguments) {
      ArrayList<String> strings = new ArrayList<String>();
      strings.add("Evan");
      strings.add("Eugene");
      strings.add("Adam");

      System.out.println(strings.size());
      System.out.println(strings.get(0));
      System.out.println(strings.get(1));

      strings.set(0, "Goodbye");
      strings.remove(1);
      for (int i = 0; i < strings.size(); i++) {
            System.out.println(strings.get(i));
       }
      for (String s : strings) {
         System.out.println(s);
      }
   }
}
```

Inheritance

Review: Classes

- User-defined data types
  - Defined using the “class” keyword
  - Each class has associated
    - Variables (any object type)
    - Methods that operate on the data (variables)
- New instances of the class are declared using the “new” keyword
- “Static” variables/methods have only one copy, regardless of how many instances are created

Example: Shared Functionality

```
public class Student {
   String name;
   char gender;
   Date birthday;
   ArrayList<Grade> grades;

   double getGPA() {
      …
   }

   int getAge(Date today) {
      …
   }
}
```

```
public class Professor {
   String name;
   char gender;
   Date birthday;
   ArrayList<Paper> papers;

   int getCiteCount() {
      …
   }

   int getAge(Date today) {
      …
   }
}
```

```
public class Person {
   String name;
   char gender;
   Date birthday;

   int getAge(Date today) {
      …
   }
}
```

```
public class Student
      extends Person {

   ArrayList<Grade> grades;

   double getGPA() {
      …
   }
}
```

```
public class Professor
      extends Person {

   ArrayList<Paper> papers;

   int getCiteCount() {
      …
   }
}
```

Inheritance

- “is-a” relationship
- Single inheritance:
  - Subclass is derived from one existing class (superclass)
- Multiple inheritance:
  - Subclass is derived from more than one superclass
  - Not supported by Java
  - A class can only extend the definition of one class

Inheritance (continued)

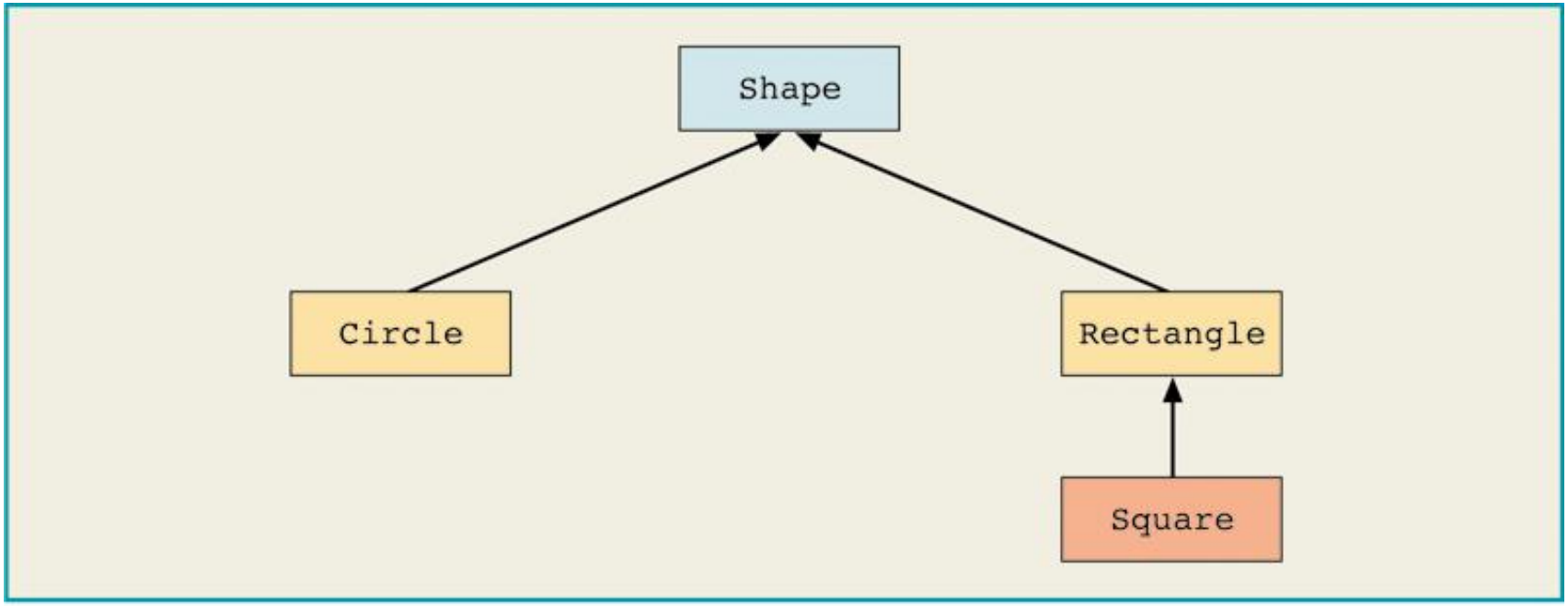

Figure 11-1 Inheritance hierarchy

```
public class ClassName extends ExistingClassName
{
    memberList
}
```

Inheritance: `class Circle` Derived from `class Shape`

```
public class Circle extends Shape
{
        .
        .
        .
}
```

Inheritance

- Allow us to specify *relationships between types*

- Why is this useful in programming?
  - Allows for code reuse
  - Polymorphism

Code Reuse

- General functionality can be written once and applied to **any** subclass
- Subclasses can specialize by adding members and methods, or overriding functions

Inheritance: Adding Functionality

- Subclasses have ***all*** of the data members and methods of the superclass
- Subclasses can add to the superclass
  - Additional data members
  - Additional methods
- Subclasses are more specific and have more functionality
- Superclasses capture generic functionality common across many types of objects

```
public class Person {
   String name;
   char gender;
   Date birthday;

   int getAge(Date today) {
      …
   }
}
```

```
public class Student
      extends Person {

   ArrayList<Grade> grades;

   double getGPA() {
      …
   }
}
```

```
public class Professor
      extends Person {

   ArrayList<Paper> papers;

   int getCiteCount() {
      …
   }
}
```

Brainstorming

- What are some other examples of possible inheritance hierarchies?
  - Person -> student, faculty…
  - Shape -> circle, triangle, rectangle…
  - Other examples???

26

UML Diagram: `Rectangle`

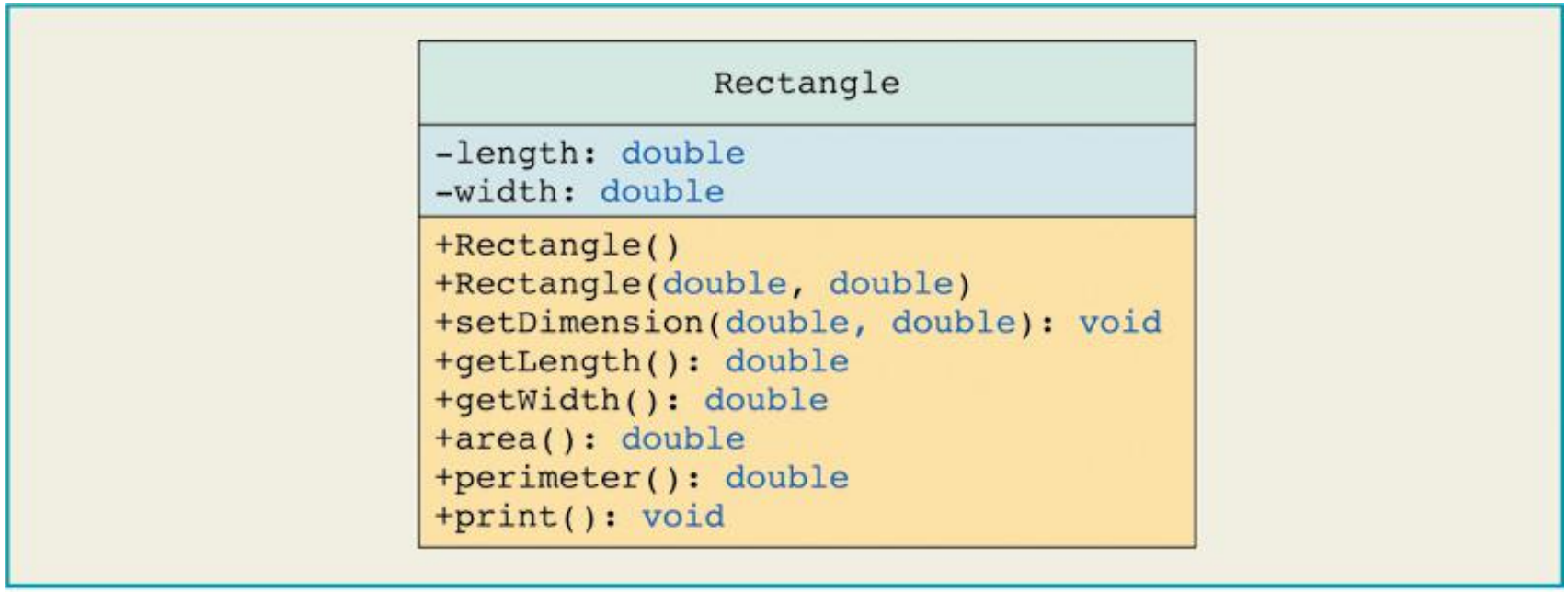

Figure 11-2 UML class diagram of the `class Rectangle`

What if we want to implement a 3d box object?

Objects `myRectangle` and `myBox`

```
Rectangle myRectangle = new Rectangle(5, 3);
Box myBox = new Box(6, 5, 4);
```

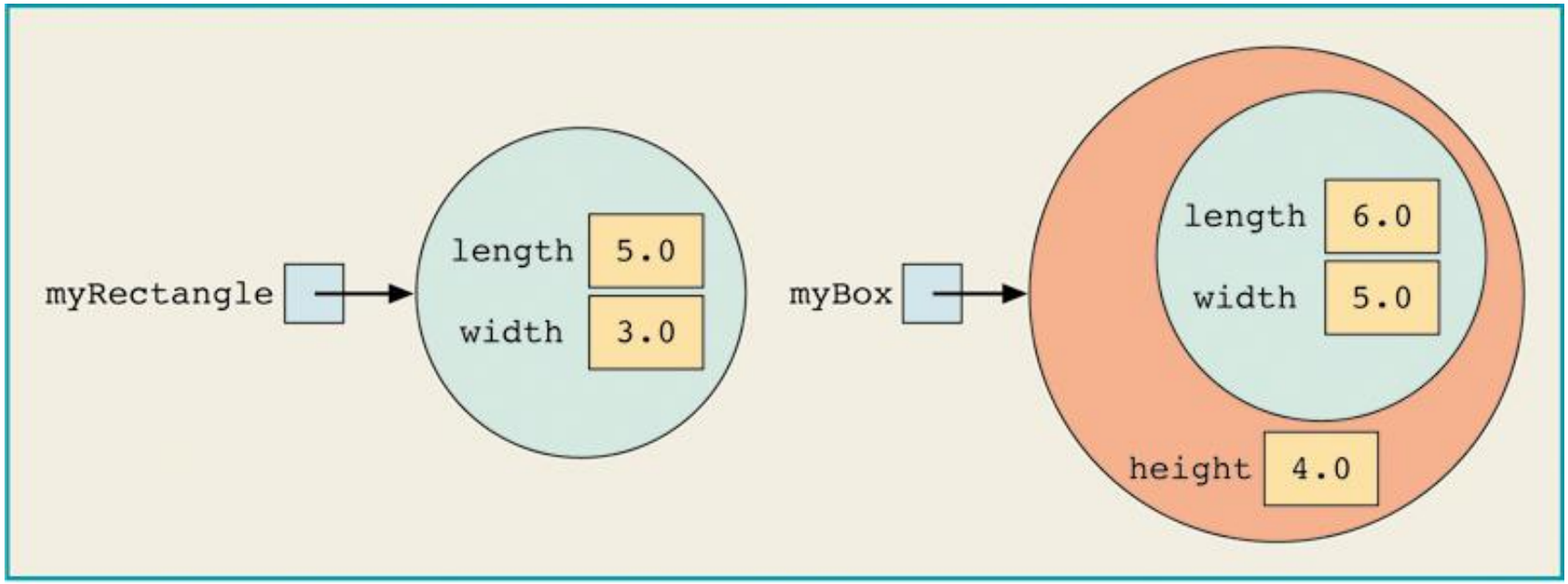

Figure 11-4 Objects `myRectangle` and `myBox`

UML Class Diagram: `class` Box

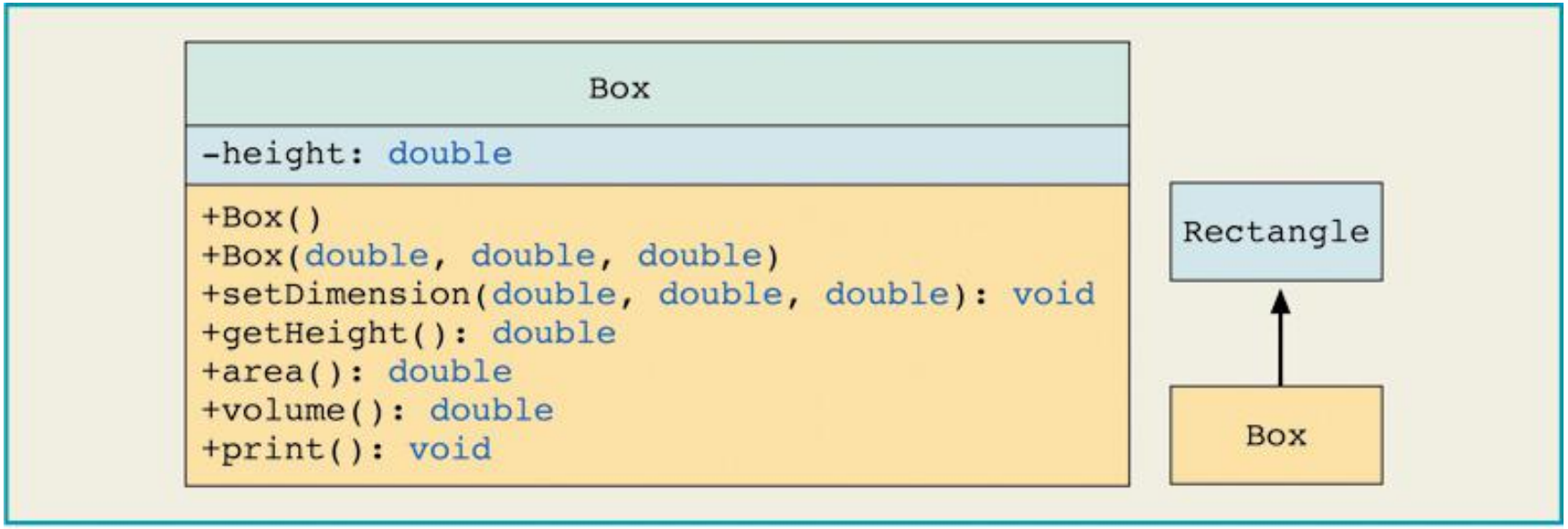

Figure 11-3 UML class diagram of the `class` Box and the inheritance hierarchy

Both a Rectangle and a Box have a surface area, but they are computed differently

Overriding Methods

- A subclass can override (redefine) the methods of the superclass
  - Objects of the subclass type will use the new method
  - Objects of the superclass type will use the original

class Rectangle

```
public double area()
{
    return  getLength() * getWidth();
}
```

class Box

```
public double area()
{
    return  2 * (getLength() * getWidth()
                 + getLength() * height
                 + getWidth() * height);
}
```

final Methods

- Can declare a method of a class final using the keyword `final`

```
public final void doSomeThing()
{
      //...
}
```

- If a method of a `class` is declared `final`, it cannot be overridden with a new definition in a derived class

Modifiers

- A subclass does not inherit/access the **private** members of its parent class.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Modifiers

- The access specifier for an overriding method can allow more, but not less, access than the overridden method.
  - a protected instance method in the superclass can be made public, but not private, in the subclass.

- You will get a compile-time error if you attempt to change an instance method in the superclass to a static method in the subclass, and vice versa.

References

- http://math.hws.edu/javanotes/
- http://www.cas.mcmaster.ca/~carette/CAS706/2004/presentations/OO_Pujari.ppt
- http://people.cs.pitt.edu/~lorym/CS401/lectures/chapter09.ppt
- http://www.cs.utexas.edu/~cannata/cs345/Class%20Notes/14%20Java%20Upcasting%20Downcasting.htm
- http://www.uwosh.edu/faculty_staff/huen/262/f09/slides/9_Abstract_Interface.ppt