CISC 3115 TY3
C06a: Visibility Modifiers and Data Encapsulation

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Concept of Java packages
- Visibility modifiers
  - public, private, and no modifiers
- Data encapsulation

Java Packages

- Programmers use packages to organize Jave classes
  - By bundling classes into packages
  - Why packages?
    - To make types easier to find and use
      - What if you created 1,000 classes?
    - To avoid naming conflicts
      - You may want to have two "Student" classes
    - To control access

Creating Package

- Use the "package" statement as the first non-comment and non-blank statement in the program

- Syntax

  package packagename

- Examples

  - package project1

  - package cisc3115.project1

  - package edu.cuny.brooklyn.cis.cisc3115.project1

Package Naming Convention

- Package names are written in all lower case to avoid conflict with the names of classes or interfaces.

Unnamed Package

- If you do not use a package statement, your type (e.g., class) is in an unnamed package.

- Generally speaking, an unnamed package is only for small or temporary applications or when you are just beginning the development process.

Package: Example

Unamed package

class Circle {

   double radius;

}

Named package

<u>package cisc3115</u>

class Circle {

   double radius;

}

Using Package Members

- Refer to the member by its fully qualified name

- Import the package member

- Import the member's entire package

Fully Qualified Name

- Fully qualified name
  - Syntax

    packagename.typename

- Example

```
class TestCircle {
    public static void main(String[] args) {
        cisc3115.Circle c1 = new Circle();
        // …
    }
}
```

package cisc3115

class Circle {

double radius;

}

Import Package Member

- To import a specific member into the current file, Use an <u>import</u> statement at the beginning of the file before any type (e.g., class) definitions but after the package statement, if there is one.

- Syntax

  import packagename.PackageMember

- Example

  ```
  import cisc3115.shapes.Circle;
  import cisc3115.shapes.Square;
  class TestShapes {
      public static void main(String[] args) {
          Circle c1 = new Circle();
          Square s1 = new Square();
      }
  }
  ```

```
package cisc3115.shapes

class Circle {
    double radius;
}

class Square {
    double length;
}
```

Import Entire Package

- To import a specific member into the current file, Use an <u>import</u> statement at the beginning of the file before any type (e.g., class) definitions but after the package statement, if there is one.

- Syntax

  import packagename.*

- Example

  import cisc3115.shapes.*;

  class TestShapes {
      public static void main(String[] args) {
          Circle c1 = new Circle();
          Square s1 = new Square();
      }
  }

package cisc3115.shapes

class Circle {
    double radius;
}

class Square {
    double length;
}

Apparent Hierarchies of Packages

- Packages appear to be hierarchical from the naming perspective, but they are not from "importing" perspective

- Example
  - cisc3115 and cisc3115.shapes are two packages when you import them.

```
package cisc3115

class Student {

    String name;

}
```

```
package cisc3115.shapes

class Circle {

    double radius;

}
```

Apparent Hierarchies of Packages: Exercise

- cisc3115 and cisc3115.shapes are two packages when you import them.

- Question: right or wrong?

```
import cisc3115.*;

class TestShapes {
    public static void main(String[] args) {
        Circle c1 = new Circle();
    }
}
```

```
package cisc3115
class Student {
    String name;
}
```

```
package cisc3115.shapes
class Circle {
    double radius;
}
```

Apparent Hierarchies of Packages: Exercise: Answer

- cisc3115 and cisc3115.shapes are two packages when you import them.

- Question: right or wrong?

```
import cisc3115.*;

class TestShapes {
    public static void main(String[] args) {
        Circle c1 = new Circle();
    }
}
```

X

```
package cisc3115
class Student {
    String name;
}
```

```
package cisc3115.shapes
class Circle {
    double radius;
}
```

Apparent Hierarchies of Packages: Example

- cisc3115 and cisc3115.shapes are two packages when you import them.

```
import cisc3115.*;
import cisc3115.shapes.*;

class TestShapes {
  public static void main(String[] args) {
    Circle c1 = new Circle();
    Student s1 = new Student();
  }
}
```

```
package cisc3115
class Student {
    String name;
}
```

```
package cisc3115.shapes
class Circle {
    double radius;
}
```

Questions?

- Concept of package
- How to name a package?
- How to import a package?
- Why do we talk about this?
  - To make types easier to find and use
    - What if you created 1,000 classes?
  - To avoid naming conflicts
    - You may want to have two "Student" classes
  - <u>To control access</u>

Visibility Modifier

- No modifier: By default, the <u>class</u>, <u>data field</u>, or <u>method</u> can be accessed by any class in <u>the same package</u>.

  - If you don't explicitly declare which package your class belongs to, the class is in the default package, an unnamed package

  - You may change this using three modifiers, public, private, and protected ("protected" to be discussed in the future) to the class, data field, or method

Public and Private Visibility Modifiers

- public

  - The class, data field, or method is visible to any class in any package.

- private

  - The data field or methods can be accessed only by the declaring class.

Visibility Modifier: Example 1

- Public and private modifiers

```
package p1;

public class C1 {
   public int x;
   int y;
   private int z;

   public void m1() {
   }
   void m2() {
   }
   private void m3() {
   }
}
```

```
package p1;

public class C2 {
   void aMethod() {
      C1 o = new C1();
      can access o.x;
      can access o.y;
      cannot access o.z;

      can invoke o.m1();
      can invoke o.m2();
      cannot invoke o.m3();
   }
}
```

```
package p2;

public class C3 {
   void aMethod() {
      C1 o = new C1();
      can access o.x;
      cannot access o.y;
      cannot access o.z;

      can invoke o.m1();
      cannot invoke o.m2();
      cannot invoke o.m3();
   }
}
```

Visibility Modifier: Example 2

- No modifiers

```
package p1;

class C1 {
   ...
}
```

```
package p1;

public class C2 {
   can access C1
}
```

```
package p2;

public class C3 {
   cannot access C1;
   can access C2;
}
```
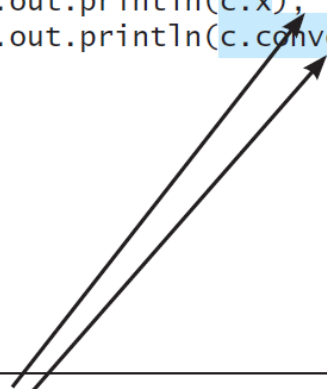
Visibility Modifier: Example 3

- Private


```
public class C {
  private boolean x;

  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }

  private int convert() {
    return x ? 1 : -1;
  }
}
```

(a) This is okay because object **c** is used inside the class **C**.

```
public class Test {
  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }
}
```

(b) This is wrong because **x** and **convert** are private in class **C**.
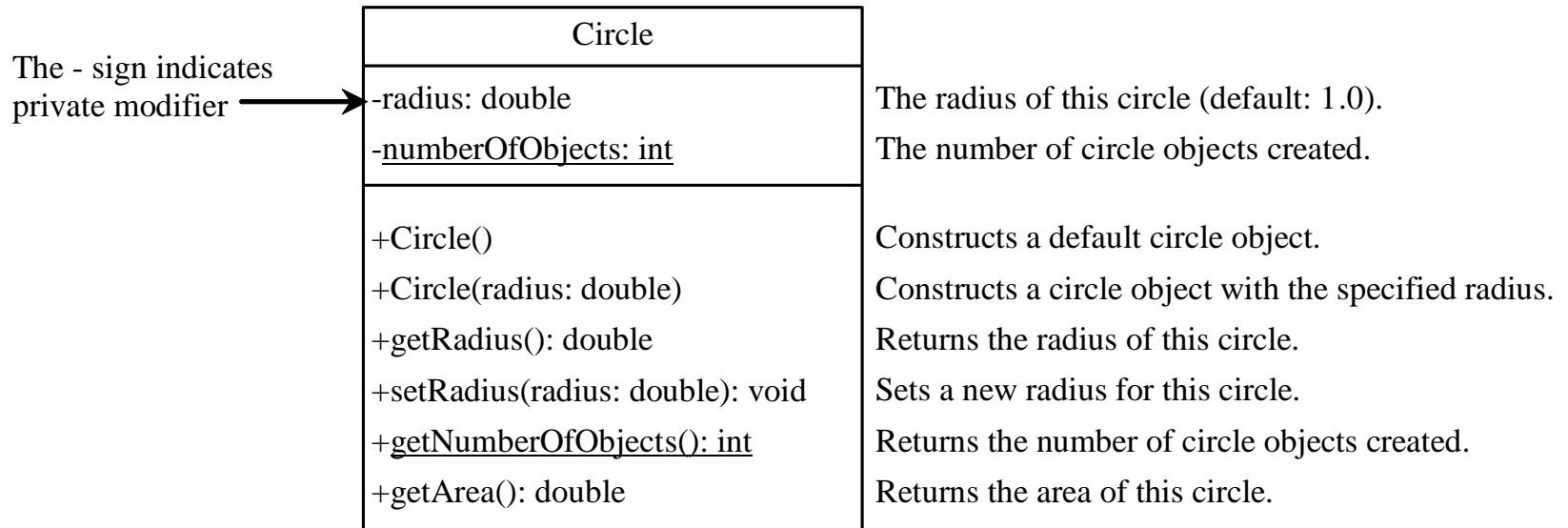
Visibility Modifiers: How to Choose?

- Consider most/more restrictive visibility modifier first, unless you have a good reason not to

  - Generally, make data fields private

  - which make code easy to maintain (why?)

Data Field Encapsulation

- <u>Making data fields private</u> protects data and makes the class easy to maintain

- Why?
  - Data may be tampered with
    - e.g., Circle.numberOfObjects = 3
  - The class becomes difficult to maintain and vulnerable to bugs
    - e.g., statement like c1.radius = -5 can be written in many places
  - Implementation also depends on the data structure

Data Field Encapsulation: Example

The - sign indicates private modifier ⟶

Circle
-radius: double
-numberOfObjects: int
+Circle()
+Circle(radius: double)
+getRadius(): double
+setRadius(radius: double): void
+getNumberOfObjects(): int
+getArea(): double

The radius of this circle (default: 1.0).

The number of circle objects created.

Constructs a default circle object.

Constructs a circle object with the specified radius.

Returns the radius of this circle.

Sets a new radius for this circle.

Returns the number of circle objects created.

Returns the area of this circle.

Questions

- Visibility modifiers
    - No visibility modifiers
    - Public and private visibility modifiers
- Data field encapsulation

CUNY | Brooklyn College

In-Class Exercise C06a-1

- In Exercise C03a-1, you have written a program that consist of two classes TV and TestTV.
  - Create a C06a-1 directory in the directory that hosts your weekly programming repository, copy the programs to the directory, and revise the program in the directory
    - Add a private static data field to the TV class and the data field is to count the number of objects of the TV class that has been created.
    - Add an instance data field to the TV class, called manufacturingDate that references a Date object represent the manufacturing date and time of aTV object.
    - Following the principle of data encapsulation, make data fields private, and make methods public
    - Test the revised program
  - Use git to make a submission