

THE SCOPE OF VARIABLES

Chapter 5, Methods, discussed local variables and their scope rules. A variable declared inside a method is called a local variable. The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. So local variables can be used only inside the method where they are declared.

Variables declared inside the class, but not inside any method in the class are called data fields or class's variables. They can be instance (non static) or static variables. Their scope is the entire class.

THE THIS REFERENCE

The this keyword is the name of a reference that refers to an object itself.

One common use of the this keyword is to reference a class's *hidden* data fields.

Another common use of the this keyword is to enable a constructor to invoke (call) another constructor of the same class.

THE THIS REFERENCE (CONTINUED)

Using the **this** keyword to reference the *hidden* data fields.

If a local variable or a parameter has the same name as a data field, then the local variable or the parameter takes precedence and the data field with the same name is *hidden* (i.e. it cannot be accessed). For example, in the following class, `i` is defined both as a data field and as a parameter in the method `setI()`. Hence the data field `i` is hidden in the method, but we can access it using the **this** keyword as shown below.

```
public class Foo {  
    private int i = 5;  
  
    public void setI(int i) {  
        this.i = i;  
    }  
}
```

The diagram illustrates the scope resolution in the provided code. It shows a class `Foo` with a private data field `i` and a public method `setI(int i)`. Inside the method, the line `this.i = i;` is highlighted. Two arrows originate from labels below: one labeled 'data field' points to the `i` in `this.i`, and another labeled 'parameter' points to the `i` in the method signature `setI(int i)`.

Suppose that `f1` and `f2` are two objects of `Foo`.

Calling `f1.setI(10)` will execute

`this.i = 10`, where **this** refers to `f1`.
The above assigns the value 10 of the parameter `i` to the data field `i`.

Invoking `f2.setI(45)` will execute

`this.i = 45`, where **this** refers to `f2`.

THE THIS REFERENCE (CONTINUED)

What if we do not use the this keyword when we should?

Since in the method `setI( )` below we did not use the `this` keyword before the `i` on the left of the assignment operator (`=`) then both the `i` on the left of `=` and on the right of `=` are treated as the parameter `i`. None of them is the data field `i`.

```
public class Foo {  
    private int i = 5;  
  
    public void setI(int i) {  
        i = i;  
    }  
}
```

The diagram illustrates that in the method `setI(int i)`, the `i` on the left of the assignment operator (`=`) is the parameter `i`. The `i` on the right of the assignment operator is also the parameter `i`, not the data field `i`. Two arrows point from boxes labeled "parameter" to the `i` on the left and the `i` on the right of the assignment operator.

Suppose that `f1` is an object of `Foo`.

Invoking `f1.setI(10)` will not initialize the data field `i` to 10. It will instead set the parameter `i` to itself (i.e. change the parameter `i` from 10 to 10).

THE THIS REFERENCE (CONTINUED)

```
public class Circle {  
    private double radius;
```

```
    public Circle(double radius) {  
        this.radius = radius;  
    }
```

```
    public Circle() {  
        this(1.0);  
    }
```

```
    public double getArea() {  
        return this.radius * this.radius * Math.PI;  
    }  
}
```

Using the this keyword to enable a constructor to call another constructor of the same class.

Here the use of the keyword this is a must to be able to access the data field radius since the parameter has the same name (radius).

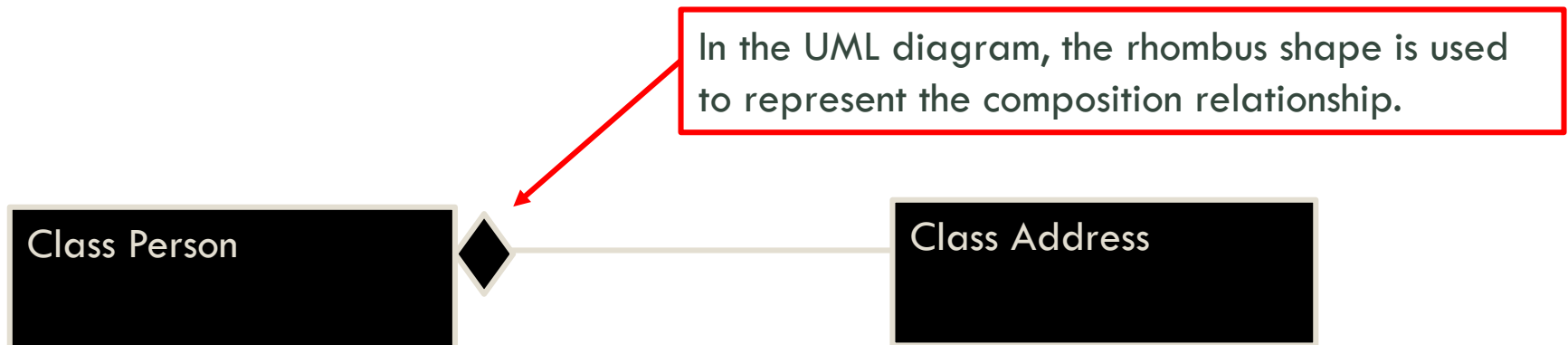
Here this is used to call the other constructor.

Here the use of this is optional since the method getArea() does not have neither a parameter nor a local variable called radius.

OBJECT COMPOSITION

An object can contain another object. The relationship between the two is called composition and is also known as the “has a” relationship.

Example:



We assume that in class Person we have a data field of type Address (Another class) which means that a person object “has an” address object (composition relationship).

OBJECT COMPOSITION (CONTINUED)



```
public class Person {
    private String name;
    private Address address; // This is the statement that creates the composition relationship.
    public Person(String n, Address a) {
        name = n;
        address = a;
    }
    public String toString() {
        return "name = " + name + " address = " + address.toString();
    }
}
```

DESIGNING THE COURSE CLASS

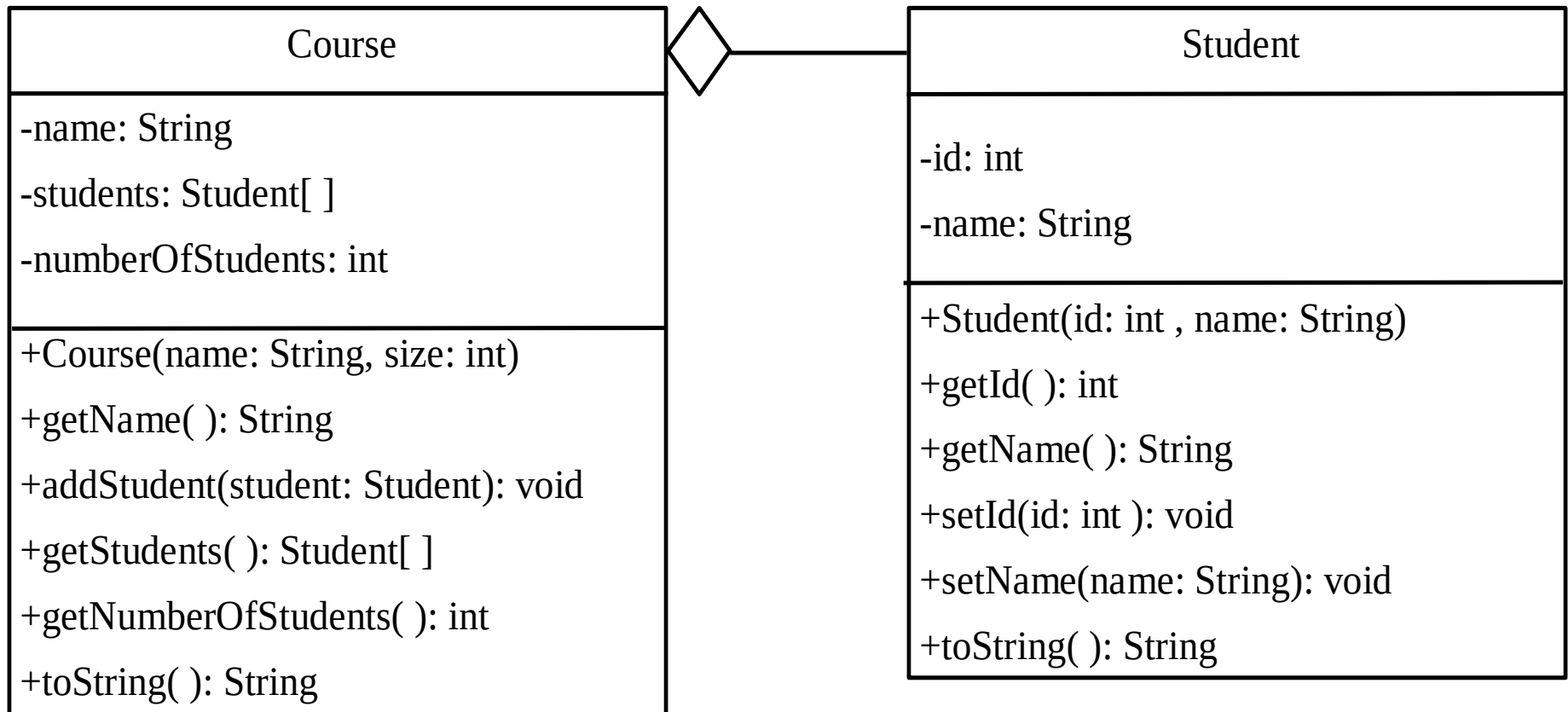
UML Diagram of class Course + description of the methods.

Course	
-name: String	The name of the course.
-students: Student[]	The students who take the course.
-numberOfStudents: int	The number of students (default: 0).
+Course(name: String, size: int)	Creates a Course with the specified name.
+getName(): String	Returns the course name.
+addStudent(student: Student): void	Adds a new student to the course list.
+getStudents(): Student[]	Returns the students for the course.
+getNumberOfStudents(): int	Returns the number of students for the course.
+toString(): String	Returns all the data fields of the class including all students' records

In your textbook the type of the array students is String, but here we changed the type to Student to make the problem more interesting.

(CONTINUED)

UML Diagram of classes Course and Student.



In your textbook the type of the array students is String, but here we changed the type to Student to make the problem more interesting.

(CONTINUED)

As you might have noticed from the UML diagram, there is a composition (has a) relationship between the class Course and the class Student. The Course class has as one of its data fields an array students of type Student. Hence a course object has many student objects.

In the next few slides you will find the code of the classes Student, Course, and an application (Driver class) to test the two classes.

In your textbook the type of the array students is String, but here we changed the type to Student to make the problem more interesting.