Department of Telecooperation

JOHANNES KEPLER UNIVERSITY LINZ

Übung Softwareentwicklung 2
für Wirtschaftsinformatik

Übung 01

Objects, Classes, and Methods

Ismail Khalil
ismail.khalil@jku.at

https://moodle.jku.at/jku/course/view.php?id=8729

Do you know?

- How to write a Java Program, compile it into byte code and run the byte code?
- How to use IDEs (Eclipse, NetBeans, IntelliJ IDEA)?
- Language fundamentals
  - Separators, Primitives, Variables, Constants, Literals, Operators
- Java statements
  - if, while, do-while, for, break, continue, switch statements
- Important classes in the Java Core Libraries
  - java.lang.Object, java.lang.String, java.lang.StringBuilder
- Basic data structures (Arrays, Lists, Sets, Maps)

NOOOOOOOOOOOO :-(

- Go back to SE1UE
- or read the Java Tutorial once again more carefully

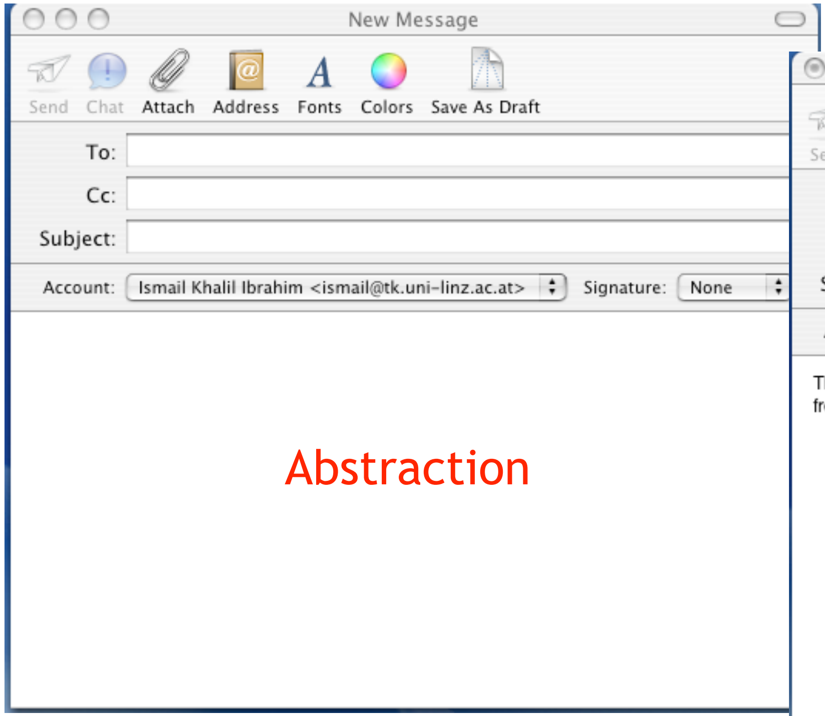
YESSSSSSSSSSSSSSSSSS :-)

- You are in the right class
- Welcome to SE2UE

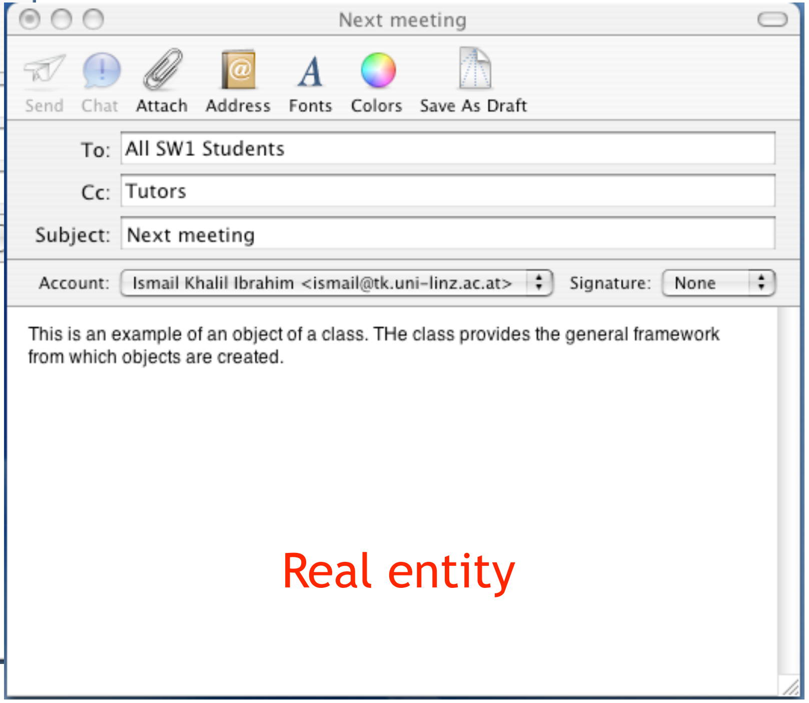
The idea of classes and objects

- A class is a model, or pattern of its objects.
  - Example

object = particular instant, or specimen of a class



Class = model, pattern



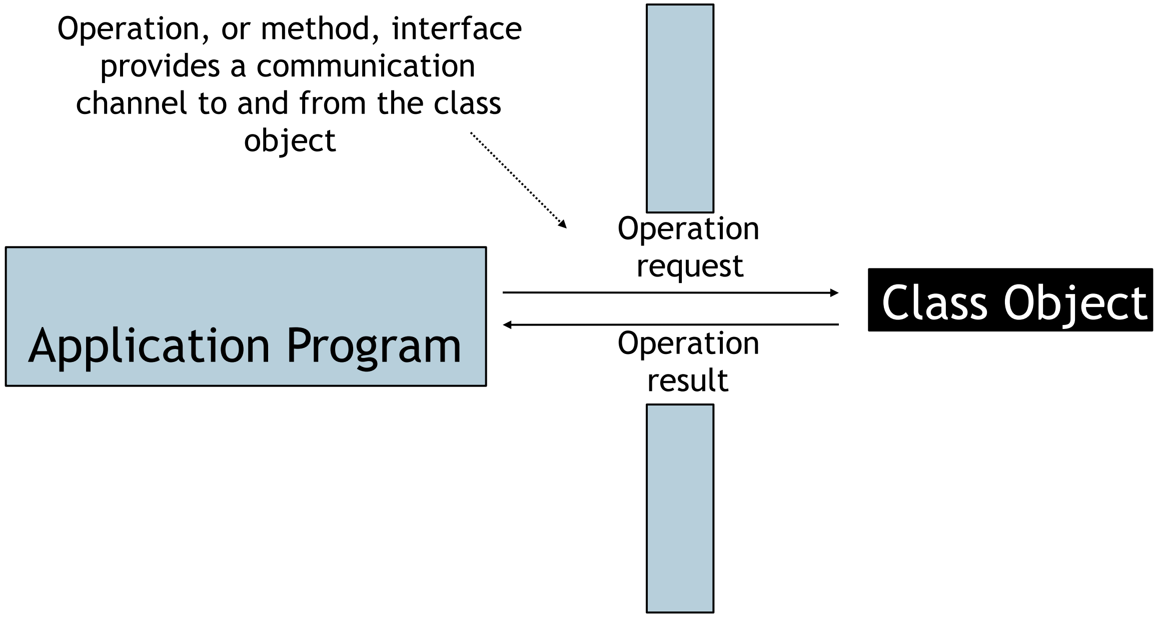
To: All SW1 Students
Cc: Tutors
Subject: Next meeting
Account: Ismail Khalil Ibrahim <ismail@tk.uni-linz.ac.at>
Signature: None

This is an example of an object of a class. THe class provides the general framework from which objects are created.

- A class provides the foundation for creating specific objects, each of which shares the general attributes, or characteristics, and behavior of the class

Classes
1. The Abstract Level

- A class is described as an interface that defines the behavior of its objects
- It provides the outside view of the class while hiding its internal structure and behavioral details


Operation, or method, interface provides a communication channel to and from the class object
Operation request
Application Program
Class Object
Operation result

Methods

- Methods are a major part of programming in Java
- Methods provides the behavior of the class and the only way to communicate with the class
  - Eliminate the need for duplicate statements
  - Make the program easier to design, code and test
  - Allow for software reusability
  - Make the program more clear and readable
  - Provides the basis to construct classes in OOP
- There are two types of methods
  - Standard, built-in, or predefined methods
    - `compareTo(), sqrt(), length()`
  - A programmer defined methods
    - A block of statements, or a subprogram, that is written to perform a specific task
- A method is given a **name** and **called**, or **invoked** using its name each time the task is to be performed within the program (**calling program**)

Void and Nonvoid Methods

- Methods can be made to serve two roles
  - nonvoid methods
    - Return a single value to the calling program
  - void methods
    - Perform specific tasks or operate on class data

Method Format

```
// METHOD SIGNATURE
modifier return_type method_name (argument_list)
{ // BEGIN METHOD BODY
   method_statement_1;
   method_statement_2;
   :
                //METHOD BODY
   :
   method_statement_n;
   return return_value;
} //END METHOD BODY
```

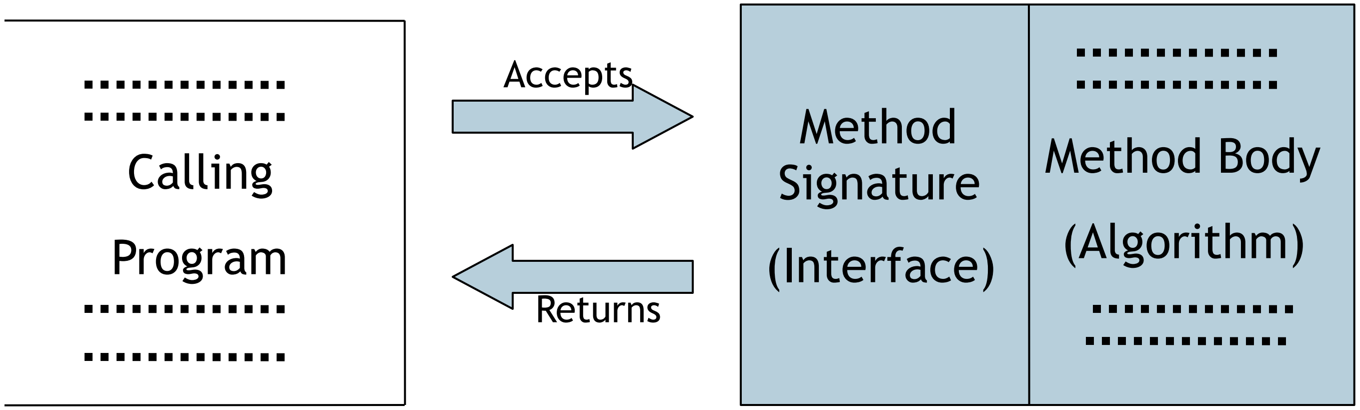
Example:

```
//THIS METHOD RETURNS THE CUBE OF AN INTEGER
static long cube(int x) // METHOD SIGNATURE
{
return x*x*x;    //METHOD BODY
}     //END cube(x)
```

The Method Signature

JOHANNES KEPLER UNIVERSITY LINZ

- The **method signature** provides the data **interface** for the method
- A method signature is a statement that forms a common boundary between the method and its calling program
- The method signature consists of four parts:
  - Optional modifier that declares the scope of the method
  - The data type of the value to be returned by the method
  - The name of the method
  - An argument list



The Modifier

- The **modifier** determines the scope of the method
- The **scope**, or **visibility** of a method defines to what extent the method can be called within the program

Modifier	Description
abstract	Body of method must appear in subclass (inheritance)
final	Can not be overridden in a subclass (inheritance)
private	Visible only in its resident class
protected	Visible to its resident class as well as any subclasses of its resident class (inheritance)
public	Visible anywhere outside its resident class (same & child classes, classes in the same or other packages)
default	visible only in its resident class and classes in the same package
static	A class method. Does not require an object to call

Static and nonstatic methods

JOHANNES KEPLER UNIVERSITY LINZ

- **static** modifier declares the method as a **class method**
- A static, or class method is a method that is not called by an object
- A class method is called by referencing its name if called within the class in which it is defined, or by using the class name and a dot operator if called outside the class in which it is defined
  - `sqrt()` in the standard `Math` class
  - `Math.sqrt(x)`
- A nonstatic method is called an instance method.
- An object is always required to call a nonstatic, or instance method
- To call an instance method, an object (instance) must be created, then this object must be used to call the method
  - `length()` in the `String` class
  - To call `length()` method, you must create a string object, say `s`, and then call the method `s.length()`

Classes
2. The Implementation Level

- At the implementation level, a class is a syntactical unit that describes a set of data and related operations that are common to its objects
- It provides the inside view of the class showing its data organization and methods implementation
- The implementation of a class consists of two data types:

1. Data members (called fields)
   - The class attributes or characteristics
   - The variables defined within the class
   - We say these members are “**private**” because they are accessible only by the method members declared for the class
2. Method members
   - Methods that operate on the data members of the class
   - These methods are usually “**public**”because they can be accessed anywhere within the scope of a given class
   - **Public** method members form the interface and exhibit the behavior of the class objects

The Class Declaration Format

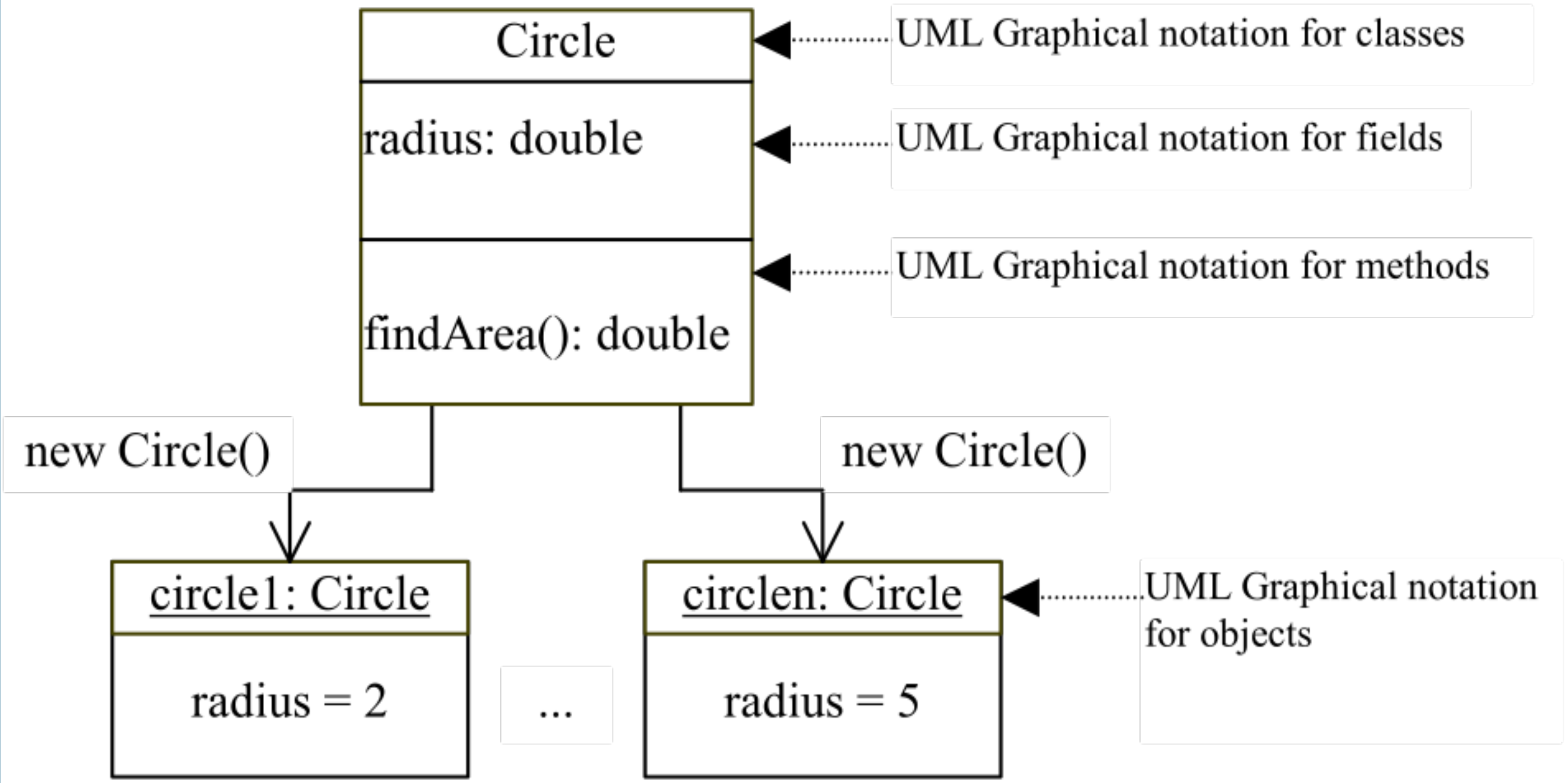
```
Class Class_Name
{
//DATA MEMBERS
private field_type variable_1;
private field_type variable_2;
:
private field_type variable_n;
//METHOD MEMBERS
public return_type method_1_name (method_1_argument_listing)
{
//METHOD 1 STATEMENTS
}
public return_type method_2_name (method_2_argument_listing)
{
//METHOD 2 STATEMENTS
}
:
public return_type method_n_name (method_n_argument_listing)
{
//METHOD n STATEMENTS
}
} // END CLASS
```

Example

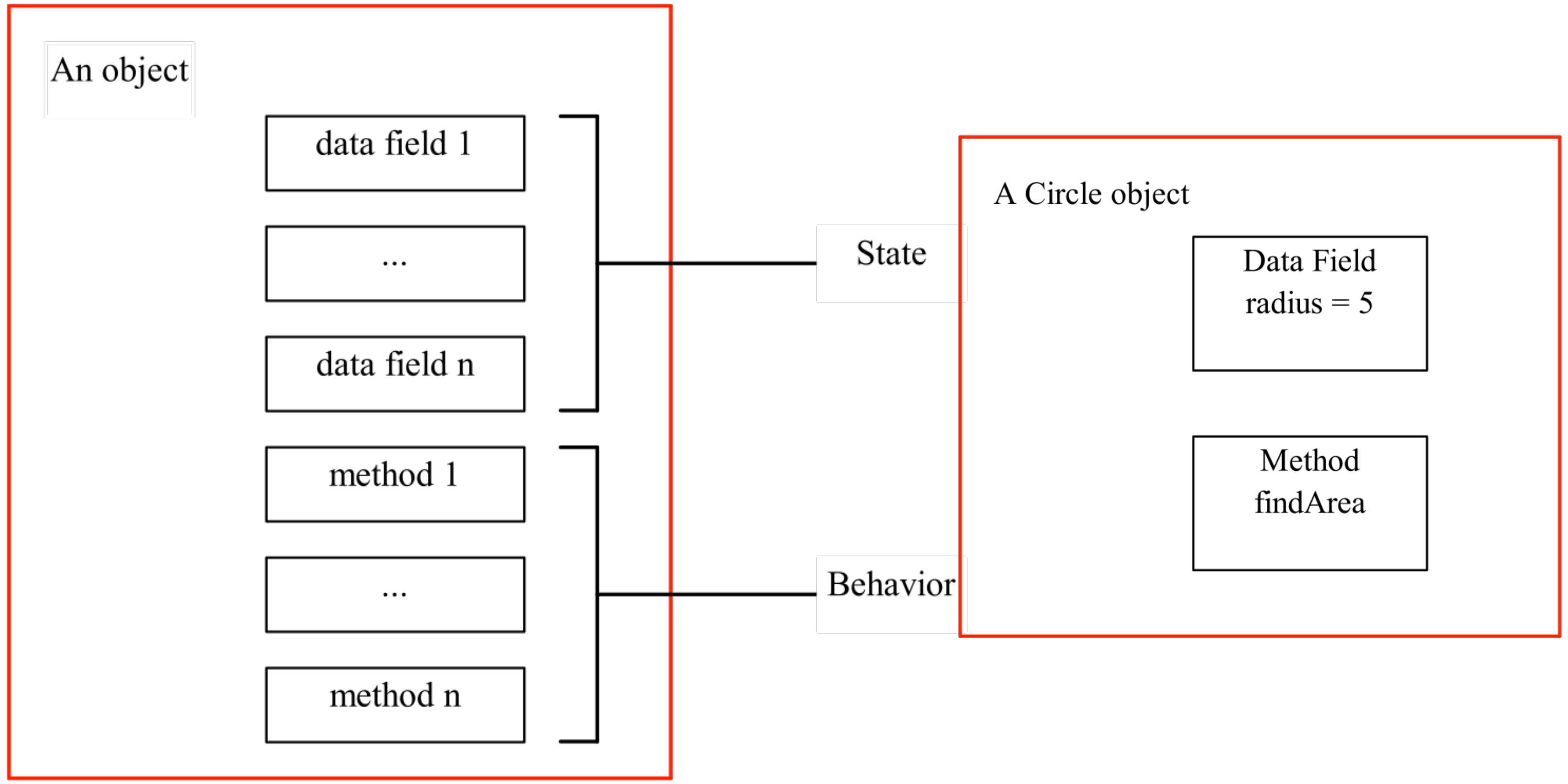
```
class Circle
{
  double radius = 1.0;

  double findArea()
  {
    return radius*radius*3.14159;
  }
}
```

Classes and Objects

- An object is an instance, or specimen of a given class
- An object of a given class has the structure and behavior defined by the class that is common to all objects of the same class


Circle
radius: double
findArea(): double
UML Graphical notation for classes
UML Graphical notation for fields
UML Graphical notation for methods
new Circle()
new Circle()
circle1: Circle
radius = 2
...
circlen: Circle
radius = 5
UML Graphical notation for objects

Objects


An object
data field 1
...
data field n
method 1
...
method n
State
Behavior
A Circle object
Data Field
radius = 5
Method
findArea

Declaring Object Reference Variables

```
ClassName objectName;
```

Example:

```
Circle myCircle;
```

Department of Telecooperation

Creating Objects

```
objectName = new ClassName();
```

Example:

```
myCircle = new Circle();
```

The object reference is assigned to the object reference variable.

Declaring/Creating Objects in a Single Step

```
ClassName objectName = new ClassName();
```

Example:

```
Circle myCircle = new Circle();
```

Accessing Objects

- Referencing the object's data:

```
objectName.data
  myCircle.radius
```

- Invoking the object's method:

```
objectName.method
   myCircle.findArea()
```

Constructors

- Every class must define a method, called a constructor which will indicate the steps, or algorithm the computer should execute when an object of this class is created or constructed
- Constructors are a special class method that is used to initialize an object automatically when the object is defined
- The name of the constructor is the same as the name of the class
- The constructor cannot have a return type, not even **void**
- Constructors should not be developed for tasks other than to initialize an object for processing

Example

```
class Circle
{
//FIELDS
private double radius=0.0;  //CIRCLE RADIUS


//METHOD MEMBERS
//CONSTRUCTOR
public Circle(double r)
{
  radius = r;
}   //END Circle()

// AREA METHOD

Public double findArea()
{
Return radius*radius*3.14159;
}  //END AREA

} // END Circle
```

Look at this Example

```
class Circle
{
//FIELDS
private double radius=0.0;   //CIRCLE RADIUS


//METHOD MEMBERS


public Circle(double radius) //CONSTRUCTOR
{
  radius = radius;
}    //END Circle()

// AREA METHOD

Public double findArea()
{
Return radius*radius*3.14159;
}   //END AREA

} // END Circle
```

The “this” Keyword

JOHANNES KEPLER UNIVERSITY LINZ

- In order to remove the ambiguity about which `radius` to be used; the `radius` as the private data member (field) or the `radius` the value received by the constructor (argument), we must tell the compiler that the `radius` on the left side of the assignment operator is the private class member (field) and the `radius` on the right side of the assignment operator is the value received by the constructor (argument) by using the `this` reference

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

The final Modifier

- A `final` class cannot be extended:

```
final class Math {
        ...
}
```

- A `final` variable is a constant:

```
final static double PI = 3.14159;
```

- A `final` method cannot be modified by its subclasses.

A word about `enum`

- Java comes with eight built-in primitive types and a large set of types that are defined by classes, such as String.
- An enum (short for enumerated types) is a type that has a fixed list of possible values, which is specified when the enum is created. In some ways, an enum is similar to the boolean data type, which has true and false as its only possible values. However, boolean is a primitive type, while an enum is not.
- The definition of an enum types has the (simplified) form:

```
enum enum-type-name {list-of-enum-values }
```

- Example

```
enum Season { SPRING, SUMMER, FALL, WINTER }
```

- Once an enum type has been created, it can be used to declare variables in exactly the same ways that other types are used.

```
vacation = Season.SUMMER;
```