Object-Oriented Programming I

Methods In-Depth

Slides by Magdin Stoica
Updates by Georg Feil

Learning Outcomes

1. Describe the role, use and definition syntax of a constructor as a special method
2. Define constructors in new and existing classes
3. Define the rules and use of method overloading
4. Overload methods in new and existing classes to improve class usability
5. Analyze categories of methods and the role of each category of methods

Reading Assignments

- Introduction to Java Programming (required)
  - Chapter 8: Objects and Classes
    - Sections 8.2, 8.3, 8.4
      Note: Examples in these sections define field variables without a visibility. This is poor programming style, we will always say public *or* private *(usually* private*) for field variables in this course*
  - Chapter 5: Methods
    - Section 5.8 (overloaded methods)
- Head First Java (required) (link is in SLATE content under General)
  - Chapter 9: Constructors and Garbage Collection
    - Read from the section "The miracle of object creation" up to and including "Nanoreview: four things to remember about constructors"

Constructors

What is a constructor?

- A constructor is a special method whose role is to initialize the object when an object is created
- The constructor is called only when an object is created ('new')
  - A constructor can't be called any other time
  - It runs at most once
- Name of a constructor is *always* the same as the name of the class, *exactly* the same
- Normally used to initialize field variables defined by the class

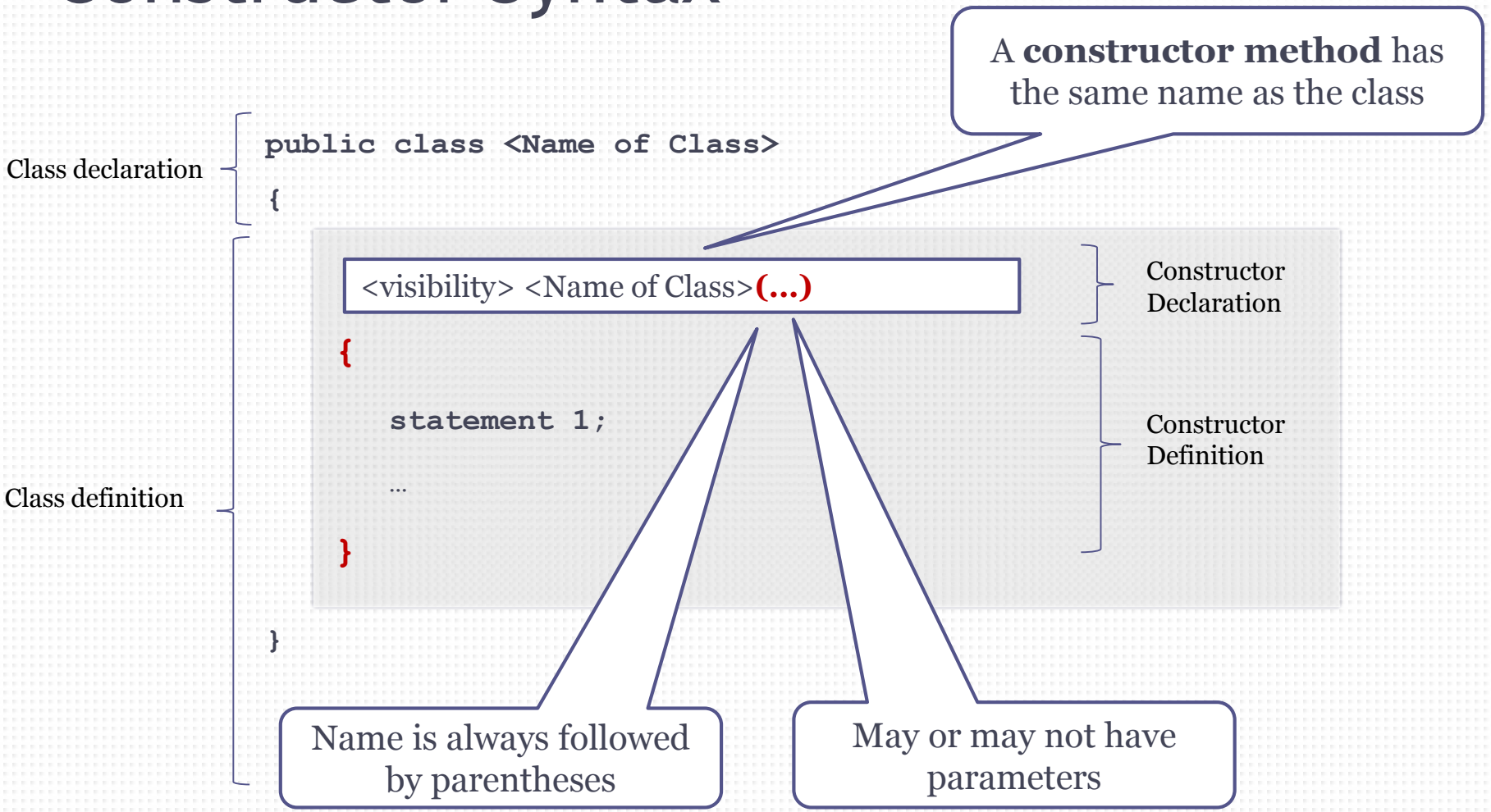
What is a constructor?

- Visibility of a constructor method is almost always 'public'
  - Can be private in advanced uses
- Return type is NOT specified because constructors can't be called in the normal way
  - Constructor methods "return" initialized objects (e.g. constructor method for a Car class will always return "Car" objects)
  - Constructors are the only methods that are allowed to omit the return type
  - Return statement with a value cannot be used (like void methods)
- Parameters are allowed but not required just like any other method

Constructor's Purpose

create and initialize
an object

Constructor Syntax



Constructor

=

Method

with

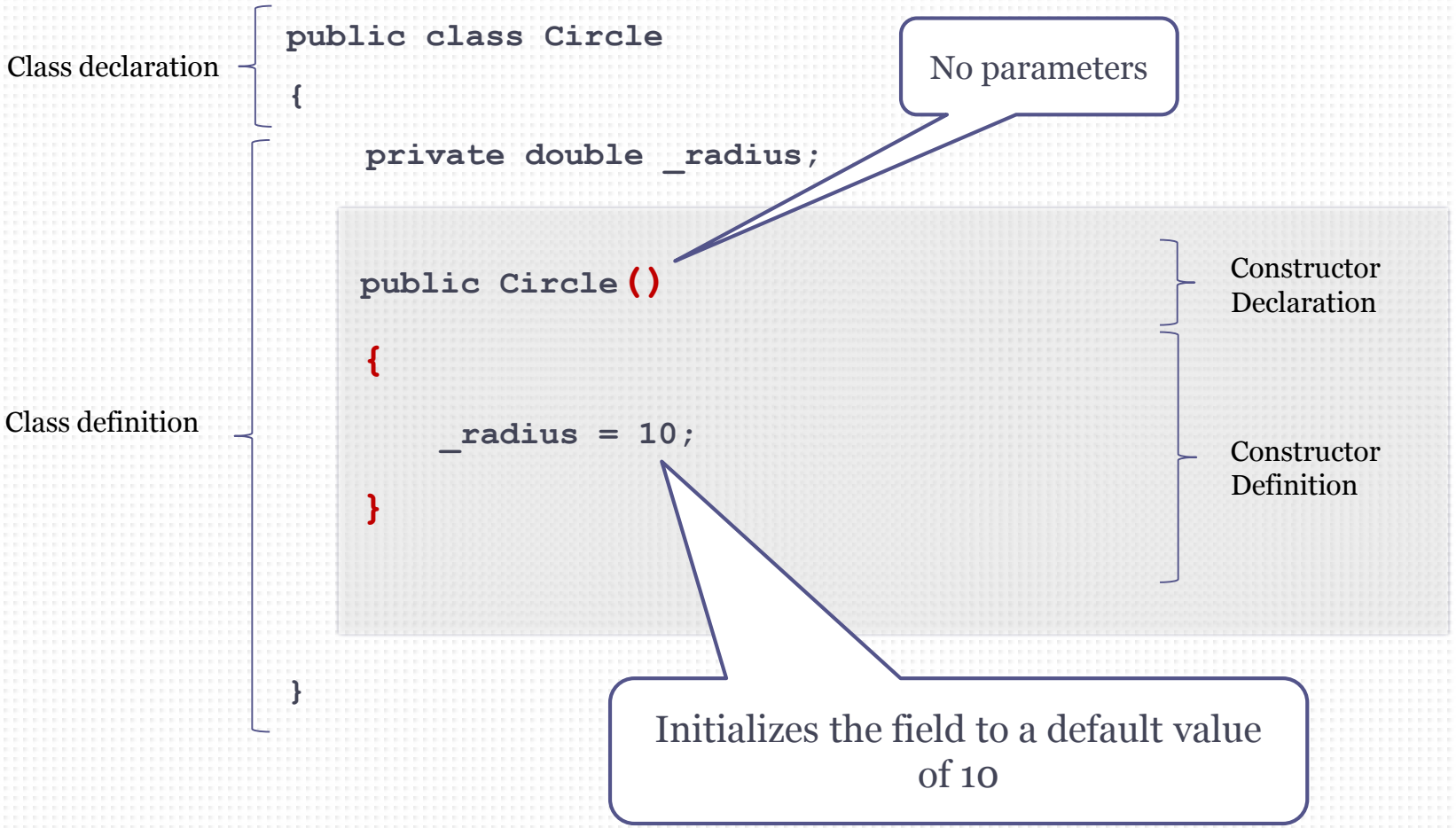
the same name as its class

and

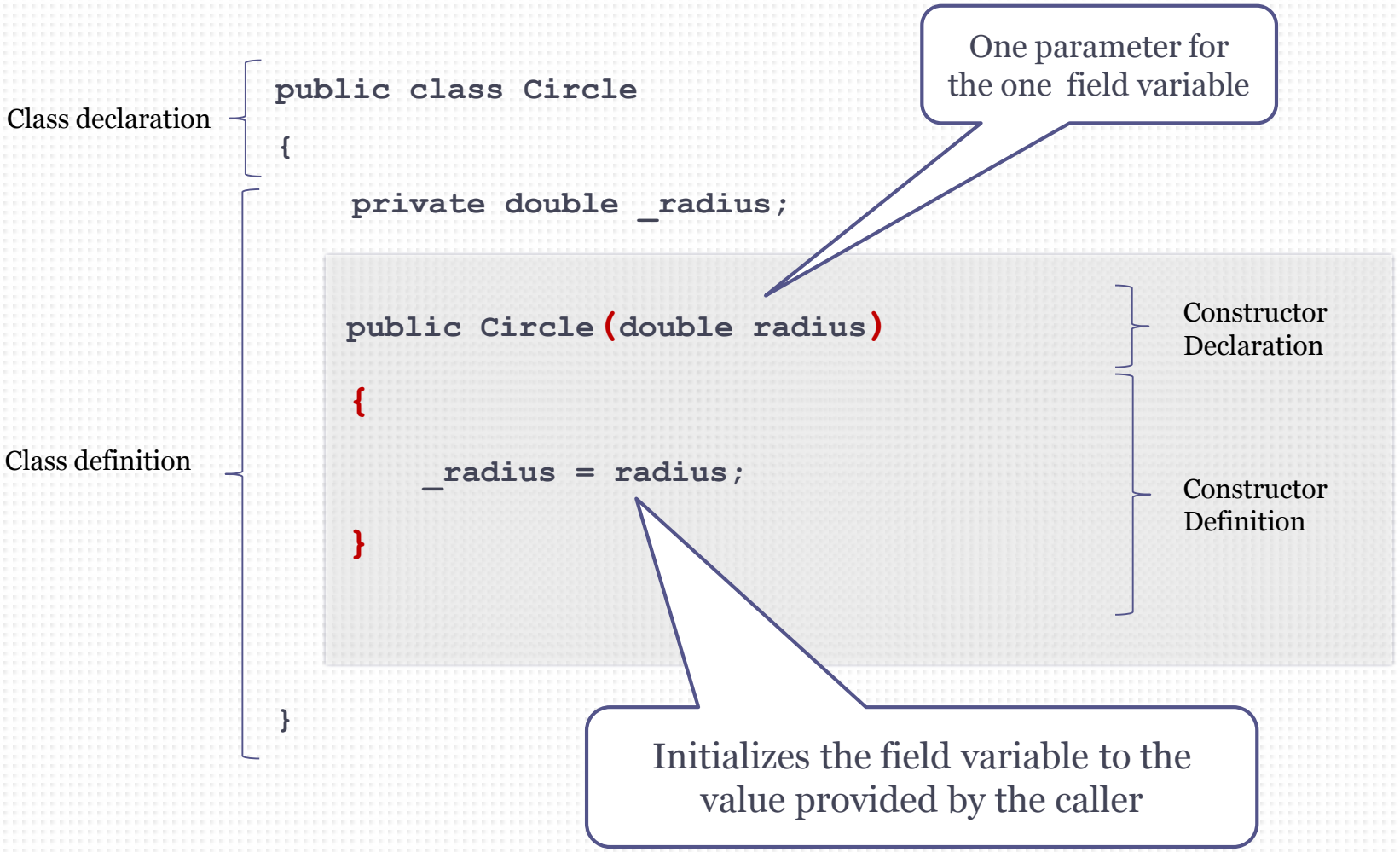
no return type

(not even ‘void’)

Example: Constructor with no parameters



Example: Constructor with one parameter



Default Constructor

- A default constructor is a constructor that has no arguments (parameters)
  - Also called no-argument constructor
- A default constructor is automatically generated if no other constructors are defined for a given class
  - If at least one constructor is defined, the default constructor is NOT automatically generated
  - If at least one constructor is defined and a default (no argument) constructor is required then you must write one
  - Automatically generated constructors don't have any "code" (empty method definition)
    - If you need a constructor to do something specific, you must write one

Calling a constructor method (pseudocode)

- Q: If the constructor is a method how do we call it?
- A: Constructors are called when an object of a class type is created using 'new'
- That's what those parentheses are for!

```
new  <class-name>();
```

Constructor name is always the same as the class name

A constructor
method call
happens when using the
new operator

Examples: Constructor Calls

- Circle defaultCircle = new Circle();
- Scanner in = new Scanner(System.in);
- Circle smallCircle = new Circle(1);
- Circle bigCircle = new Circle(1000);
- Rectangle rect = new Rectangle(100, 30);
- Employee president = new Employee("Barack Obama");
- Car smartCar = new Car("Chevy Volt");

Constructors vs. Methods

Constructors

- Name is always the same as the class name
- Return type is not specified, not even using void
- The return statement with a value is not allowed
- Called when using the "new" keyword
- The constructor call always creates a new object
- Can have any number of parameters

Regular Methods

- Can have any desired name (start with lower-case letter)
- Return type must be specified or "void" must be used
- The return statement may be used to return a value
- Invoked through simple method call statement
- Can only be called if the object exists (or method is static)
- Can have any number of parameters

Be careful!

- What will Java do with this?

```
public class Database {
    public void Database() {
        // Initialization goes here
    }
}
```

- *This is allowed*, gives no compile error
  - Creates a default constructor as usual
- Java treats this as a normal method, not a constructor (but it shouldn't… this should be an error… so never do this)

Method Overloading

Method Overloading

- Overloading means more than one method with the same name
- Any method can be overloaded
- The method name must be the same
- The list of parameters must be different in each method
  - You (and the compiler) can tell which version of the method is being called by what parameters are being passed
- The return type can be different but doesn't have to be
- Different method implementations with the same name should all have a similar meaning / purpose
  - Similar purpose achieved using different input

Method Overloading Examples

- From your textbook, Chap 5 pg. 194

```
public int max(int num1, int num2) {…}
public double max(double num1, double num2) {…}
public int max(int num1, int num2, int num3) {…}
```

- Also, see how println() is defined in the Java library!
  - Look up the PrintStream class on javadocs.org

Method Overloading Examples

- Here is how the max() methods might be implemented

```
// Returns the highest integer parameter value
public int max(int num1, int num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```

Method Overloading Examples

```
// Returns the highest double parameter value
public double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```

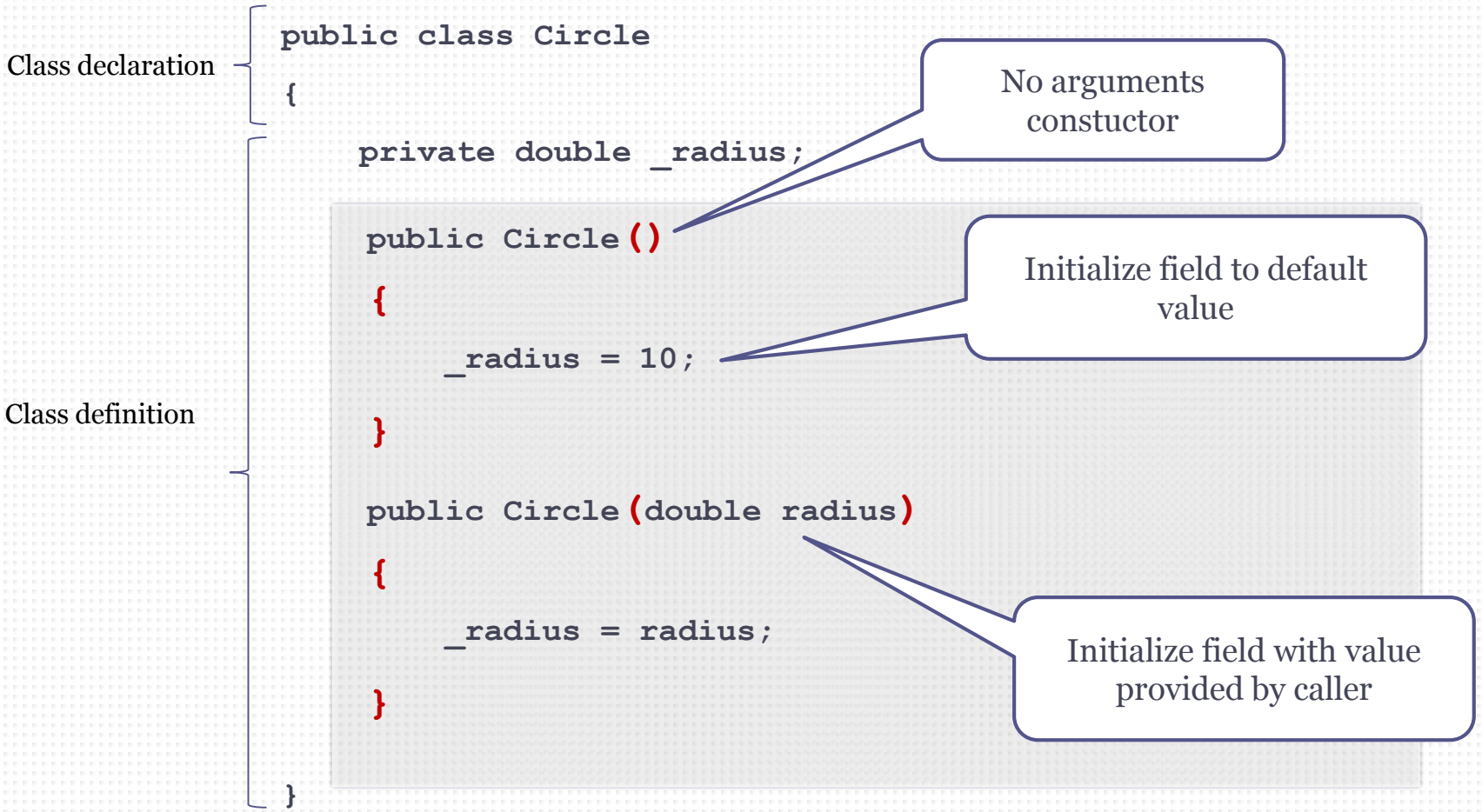
Method Overloading Examples

```
// Returns the highest integer parameter value
public int max(int num1, int num2, int num3) {
    return max(max(num1, num2), num3);
}
```

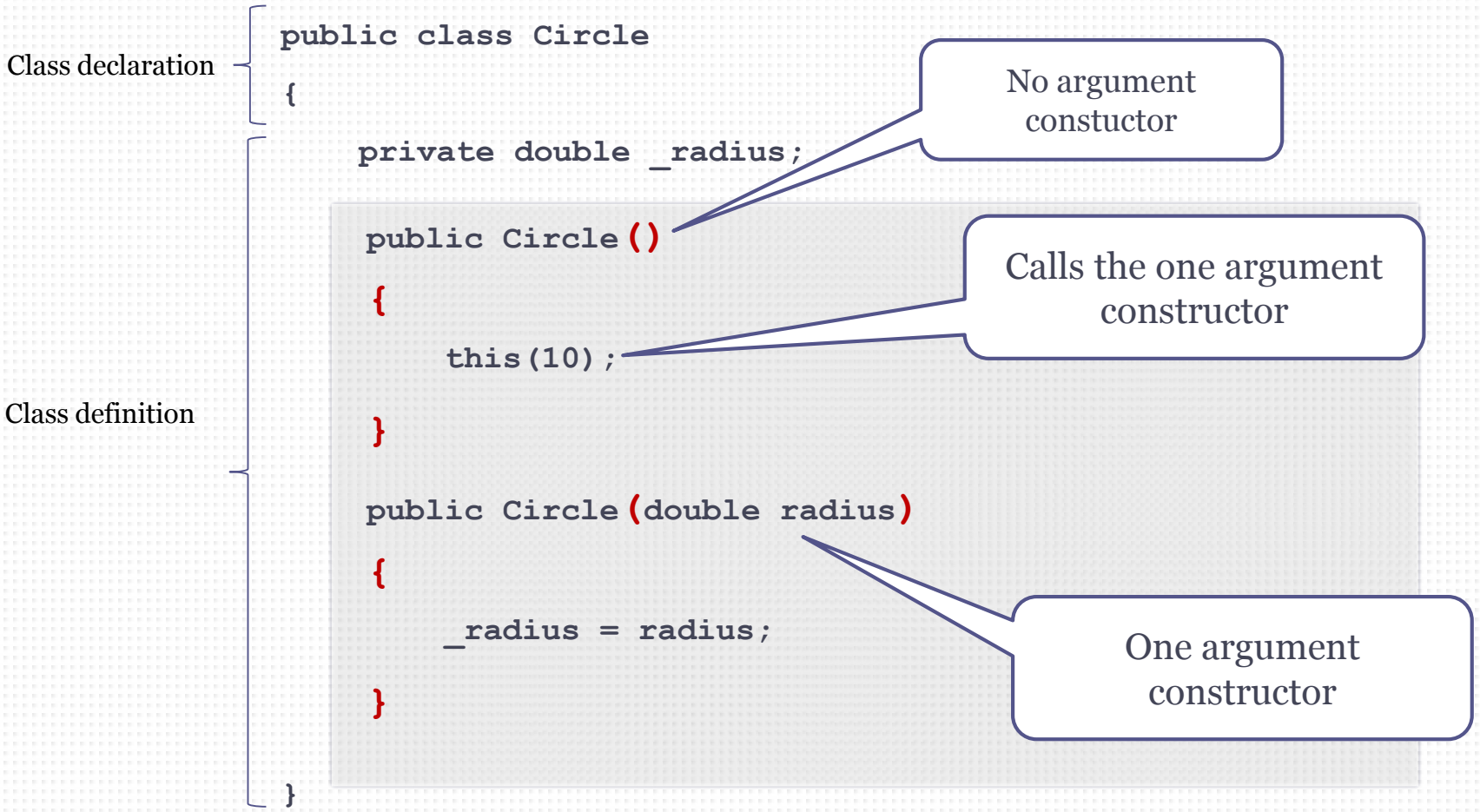
Constructor Overloading

- Constructors are just a special kind of method, so they can be overloaded if needed
- A class can have many different constructors
  - Name is always the same as the class name
  - Parameter list is different for each overloaded constructor
- It is very common to overload constructors
  - Create the same kind of object with different inputs

Example: Constructor Overloading



Example: Constructor Overloading



Exercise 1: Barking dogs with constructors

- Start with the latest barking dogs example (Dog2 and DogTest classes)
- Define two constructors for the Dog2 class
  - Default (no-argument) constructor
  - Two-argument constructor to define the dog's name and size
- Implement the default (no argument) constructor by calling the two-argument constructor with default values
- Update the DogTest class to create Dog2 objects using the two-argument constructor instead of the mutators (setters)
  - You can still use the mutators later to change the initial value

Non-Java Concept: Destructors

- Just like a *constructor* is called when an object is created, some OO languages (e.g. C++) have the concept of *destructor*
  - Called when the object is destroyed (deallocated)
- Java does not have destructors
- If in more advanced Java programming you need to run some cleanup code when an object is deallocated, define the method

```
protected void finalize()
```

- This gets called just before the JVM deallocates your object (not recommended, use only if really needed)

Summary: Steps to implement a class

1. Declare and define the class (empty definition for now)
   - Save and compile to ensure everything is defined correctly.
2. Define the field variables *that you know about* (not too many)
3. Define all necessary constructors
   - Initialize all field variables in each constructor in the order they are defined (the right order helps you be consistent)
4. Define all necessary accessor and mutator methods
   - Not all fields need them but many do.
5. Define other methods that implement the functionality required by the instances of the class, the objects (from s/w requirements)
   - Don't write more methods than you need
6. Iterate, starting again at step 2. Do you need more fields, constructors, getters/setters, or other methods?