Constructors and Encapsulation

COP2250: Java Programming

Kevin Pyatt, Ph.D.

State College of Florida
Pyatt Labs

Week 3

Today's Objectives

- Understand the problem with public fields
- Create constructors to initialize objects
- Use `private` fields for encapsulation
- Write getter and setter methods
- Understand the `this` keyword

Week 2 Review: The Problem

Last week's approach:

```
Pet pet1 = new Pet();
pet1.name = "Buddy";
pet1.age = 3;
pet1.speak();
```

What's wrong with this?

- What if someone forgets to set `name`?
  Output: "null says: Woof!"
- What if someone sets `age = -5`?
  Nothing stops them.
- Fields are **exposed** — anyone can change them.

The Solution: Constructors + Encapsulation

Week 2 (Fragile):

- Public fields
- Manual initialization
- No validation
- Easy to break

Week 3 (Robust):

- Private fields
- Constructor initialization
- Controlled access
- Hard to misuse

What is a Constructor?

A **constructor** is a special method that:

- Has the **same name** as the class
- Has **no return type** (not even void)
- Runs **automatically** when you call `new`
- Initializes the object's fields

```java
public Pet(String name, int age, String type) {
    this.name = name;
    this.age = age;
    this.type = type;
}
```

Using Constructors

Before (3 lines, easy to forget):

```
Pet pet1 = new Pet();
pet1.name = "Buddy";
pet1.age = 3;
```

After (1 line, guaranteed complete):

```
Pet pet1 = new Pet("Buddy", 3, "Dog");
```

Key insight: Constructor *forces* you to provide required data upfront.

The this Keyword

Problem: Parameter name matches field name.

```
public Pet(String name, int age, String type) {
    name = name; // Which name? Confusing!
}
```

Solution: Use this to refer to the object's field.

```
public Pet(String name, int age, String type) {
    this.name = name; // this.name = field
    this.age = age;   // name = parameter
    this.type = type;
}
```

this means "this object's" field.

Private Fields

Public fields — anyone can access and modify:

```
public String name;  // Anyone can do pet1.name = "X"
public int age;      // Anyone can do pet1.age = -5
```

Private fields — only this class can access:

```
private String name; // pet1.name = "X" --> ERROR
private int age;     // pet1.age = -5 --> ERROR
```

Encapsulation: Hide the data, control access through methods.

Getters: Read Access

A **getter** returns the value of a private field.

```java
private String name;

public String getName() {
    return name;
}
```

Usage:

```java
System.out.println(pet1.getName()); // "Buddy"
// pet1.name --> ERROR (private)
```

Convention: get + FieldName (camelCase)

Setters: Write Access with Validation

A **setter** modifies a private field — with control.

```
private int age;

public void setAge(int age) {
    if (age >= 0) {        // Validation!
        this.age = age;
    }
}
```

Usage:

```
pet1.setAge(5);  // Works
pet1.setAge(-5); // Ignored (validation fails)
// pet1.age = -5 // ERROR (private)
```

Convention: `set` + FieldName (camelCase)

Pet.java — Complete

```java
public class Pet {
    private String name;
    private int age;
    private String type;

    public Pet(String name, int age, String type) {
        this.name = name;
        this.age = age;
        this.type = type;
    }

    public String getName() { return name; }
    public int getAge() { return age; }
    public String getType() { return type; }

    public void setAge(int age) {
        if (age >= 0) { this.age = age; }
    }

    public void haveBirthday() {
        age++;
        System.out.println(name + " had a birthday!");
    }
}
```

PetShop.java — Using the Class

```java
public class PetShop {
    public static void main(String[] args) {
        // Create with constructor
        Pet pet1 = new Pet("Buddy", 3, "Dog");
        Pet pet2 = new Pet("Luna", 5, "Cat");

        // Use getter
        System.out.println(pet1.getName());

        // Use method that modifies state
        pet1.haveBirthday();

        // Use getter to see change
        System.out.println(pet1.getName() + " is now "
            + pet1.getAge() + " years old.");
    }
}
```

Summary: Before vs After

Concept	**Week 2**	**Week 3**
Fields	`public`	`private`
Initialize	Manual, 3 lines	Constructor, 1 line
Read data	`pet1.name`	`pet1.getName()`
Write data	`pet1.age = x`	`pet1.setAge(x)`
Validation	None	In setter
Safety	Fragile	Robust

Key Terms

Constructor	Special method that initializes an object
Encapsulation	Hiding data, exposing controlled access
Private	Access modifier — only this class can see it
Getter	Method that returns a field's value
Setter	Method that modifies a field's value
this	Refers to the current object's field

Lab: Pet Challenge Part 2

Task: Upgrade your Pet class from Week 2.

1. Add `private` fields: name, age, type
2. Add constructor: `Pet(String, int, String)`
3. Add getters: `getName()`, `getAge()`, `getType()`
4. Add setter: `setAge(int)`
5. Add method: `haveBirthday()`
6. Update `speak()` to use type for different sounds

Repo: `https://github.com/PyattLabs/scf-java-labs`
`labs/pet-challenge-part2/`

Next Steps

- Complete Pet Challenge Part 2
- Push to GitHub
- Review: constructor, private, getter, setter, this
- **Next week**: Static methods and class variables

Be the engineer who can fix it.

COP2250: Java Programming

Kevin Pyatt, Ph.D.

State College of Florida