Encapsulation

Java Developer

StayAhead Training; January, 2023

Encapsulation and Data Hiding

Encapsulation

- Encapsulation is the grouping together of related state and behaviour
- A class encapsulates related fields and methods
- The fields of a class ought to be related and the methods ought to operate on those fields - if that's not the case then the class is not well encapsulated
- Data hiding falls under the umbrella of encapsulation
- Data hiding is the preventing of direct access to an object's state
- The internal workings of a class should also be hidden

Access Modifiers

Encapsulation

- An access modifier is a keyword that dictates which objects can access a given class, field, or method

• private	can only be accessed by this class
• <none>	can be accessed by any class in this package
• protected	package access + subclasses in any package
• public	can be accessed by any class in any package

- Note that when we say 'can be accessed by this class/any class' what we really mean is *instances* of this/any class

Access Modifiers

Encapsulation

- privateInt is accessible to class A only
- packageInt is accessible to classes A and B
- protectedInt is accessible to classes A, B, and D
- publicInt is accessible to classes A, B, C, and D

package 1	package 2

public class A
private int privateInt;
int packageInt;
protected int protectedInt;
public int publicInt
	public class C

public class B	public class D extends A

private Fields

Encapsulation

- An object whose fields are accessible directly is prone to corruption/ invalidation, e.g.

```
var account = new BankAccount();
account.balance = 1_000_000;
```

- To prevent this, fields ought to be made private
- Those fields that must be read and/or written to by instances of other classes must have accompanying methods that are accessible from outside

public Methods

Encapsulation

- private fields are often accompanied by public methods
- More generally, public methods represent the class's interface/API
- Not all class methods should be public
- Those methods that are called only by other methods of the class should not be public (the internal workings of a class should be hidden)
- Depending on the way you structure your classes into packages, methods might suffice having package access (no modifier)

Getters and Setters

Encapsulation

- A getter/accessor method is one that returns a copy of a private field
- Convention dictates that the method name is the field name in title case and prefixed with 'get'

```
private String email;

public String getEmail() {
  return email;
}
```

Getters and Setters

Encapsulation

- A setter/mutator method is one that writes to a private field
- Convention dictates that the method name is the field name in title case and prefixed with 'set'

```
private String email;

public void setEmail(String email) {
  this.email = email;
}
```

Getters and Setters

Encapsulation

- Consider the following setter method:

```
public void setEmail(String email) {
  this.email = email;
}
```

- As the parameter name matches the field name the this keyword must be used to distinguish between the field and the parameter
- The parameter need not share its name with the field but doing so is conventional - your editor will generate code like this automatically

Getters and Setters

Encapsulation

- Setter methods don't implicitly protect objects from corruption/invalidation; to do that they must include some business logic
- The setEmail method, for example, may include code to test the email parameter against a regex before setting the field
- A setter method is likely to throw an exception if the input data is invalid
- Validation is not the only reason for coding setter methods, however

The Java Bean Specification*

Encapsulation

- The Java Bean Specification is a set of requirements relating to the way Java classes are structured
- A class that adhere to the spec. is a Java Bean
- The spec. typically applies only to those classes whose instances represent the app's data, not the classes that do the app's work
- An app whose classes conform to the spec. enable introspection by other applications, tools, and frameworks

The Java Bean Specification*

Encapsulation

- The Java Bean Specification requires that the class has:
    - private fields
    - A constructor that accepts no arguments
    - public getter and setter methods for each field
- Hibernate is a Java library that simplifies the reading objects from and writing objects to a relational database - it expects classes to adhere to the Java Bean Specification by default

Constructors (a review)

Encapsulation

- A constructor is like a method and is called with the new keyword to instantiate the class/create an object of the class
- Its name must match the class name
- It must not return anything or specify a return type
- If you do **not** add one to your class then the compiler will add one for you with no parameters (a no-args constructor)
- If you do add one to your class then the compiler will **not** add one for you
- Constructors, like methods, may be overloaded

Constructors

Encapsulation

- Constructors can help to prevent the creation of corrupt/invalid objects
- Constructors specify the legit. ways in which the class may be instantiated
- That is, you can force the caller to provide certain data
- Rather than setting fields directly, the constructor might call setter methods to ensure business rules are enforced
- A constructor is likely to throw an exception if the input data is invalid

The this Keyword (a review)

Encapsulation

- In a class the this keyword references the current object; consider…

```
var book1 = new Book("My Book", "Stuart");
```

- The new keyword results in the creation of an empty object
- The empty object is referenced in the constructor using this, e.g.

```
this.title = title;
```

- The code in a class operates on some object that does not exist until runtime; the this keyword is used to reference that object

Records (since Java 14)

Encapsulation

- An immutable object is one that cannot be changed
- Prior to Java 14, a class without setter methods yields immutable objects, e.g.

```
public class Client {
  private String name;
  public Client (String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
}
```

Records (since Java 14)

Encapsulation

- A record is a class whose instances are immutable e.g.

```
public record Client(String name) {}

var client = new Client("Smith");

var name = client.name(); // Smith
```

- Note that the record's getter method is named name, not setName
- It is possible to add methods to a record though it is not recommended

Records (since Java 14)

Encapsulation

- A record's constructor may be customised e.g.

```java
public record Client(String name) {
  public Client {
    if (name == null || name.isBlank()) {
      throw new IllegalArgumentException("Name is reqd");
    }
  }
}
```

- Note that you need not specify constructor parameters