Modifier, Encapsulation, and Class Relationship

Object-oriented Programming

Aryo Pinandito, S.T., M.MT, Ph.D.

Modifier

Visibility Modifier

- By default, the class, variable, or method can be accessed by any class in the same package.
- Public—The class, data, or method is visible to any class in any package.
- Private—The data or methods can be accessed only by the declaring class.

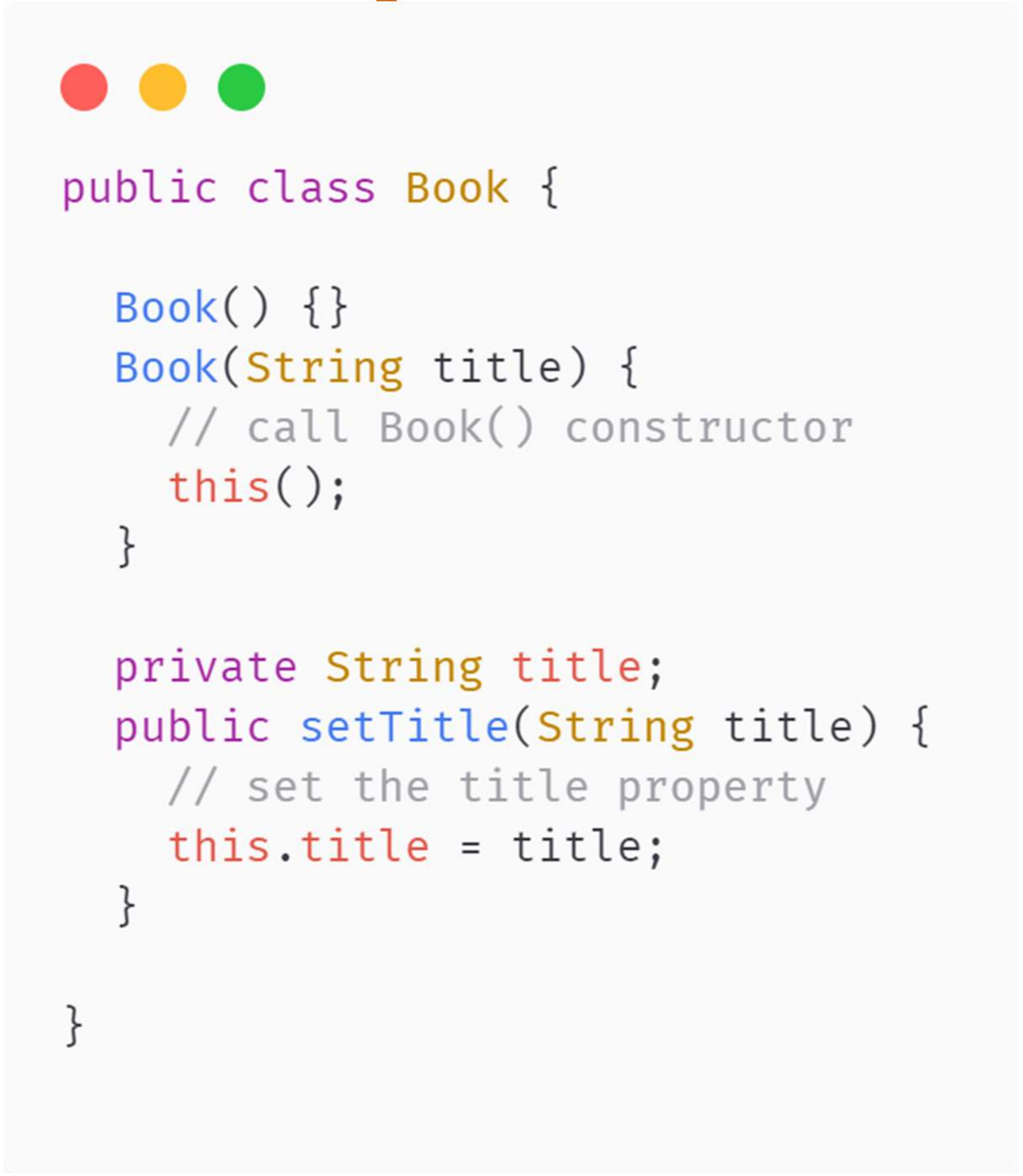
The this Keyword

- The this keyword is the name of a reference that refers to an object itself.
  - One common use of the this keyword is reference a class’s hidden data fields.
- Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.

this Keyword Example

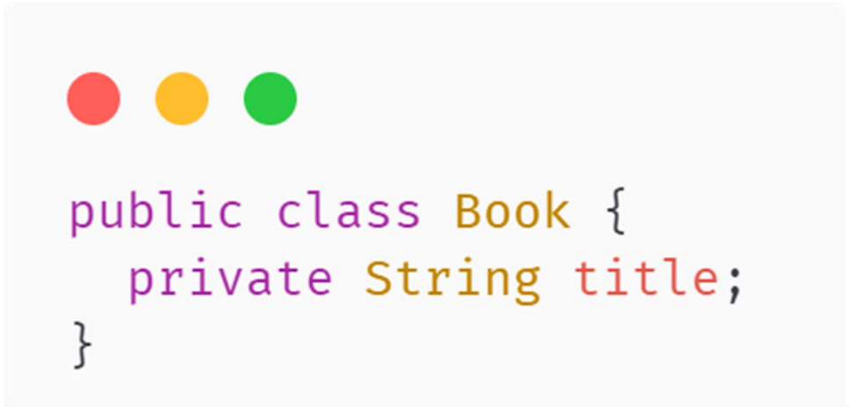
- Invoking (calling) another constructor

- Accessing attribute of the same name with a method parameter

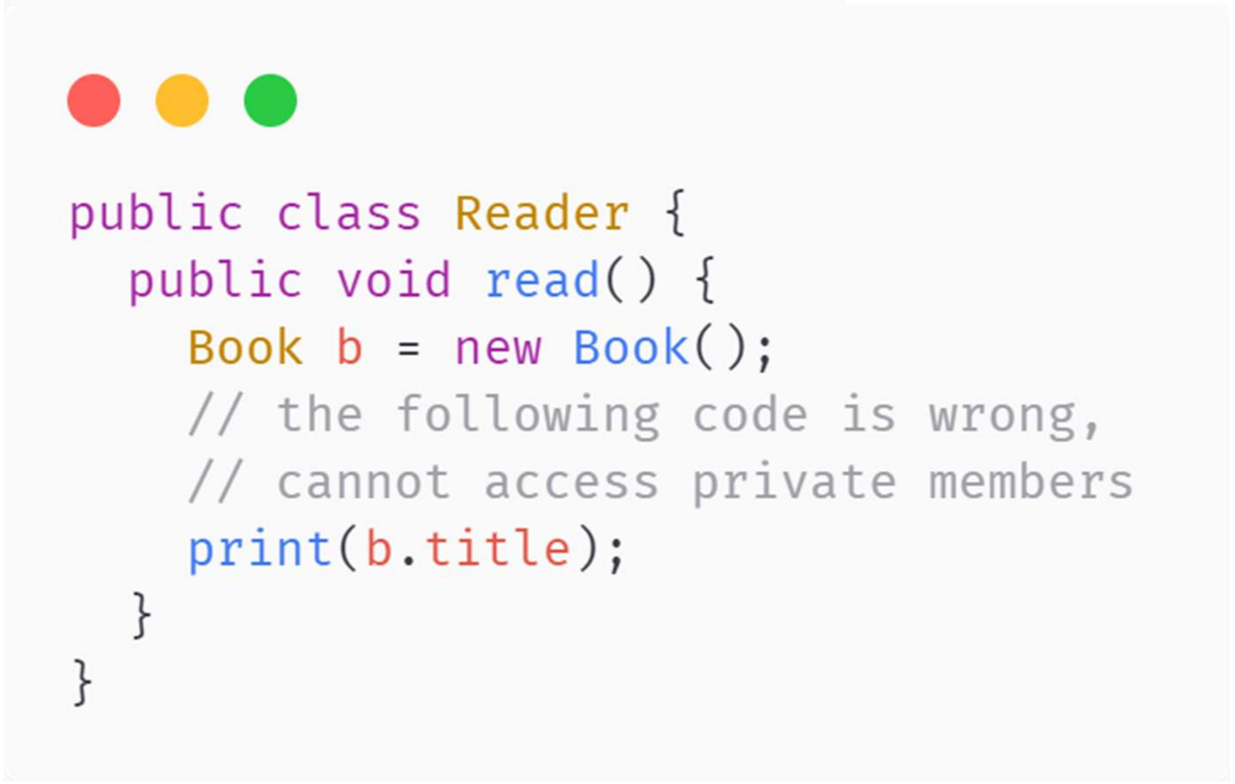


```java
public class Book {

  Book() {}
  Book(String title) {
    // call Book() constructor
    this();
  }

  private String title;
  public setTitle(String title) {
    // set the title property
    this.title = title;
  }

}
```

Private Modifier Example



```java
public class Book {
  private String title;
}
```

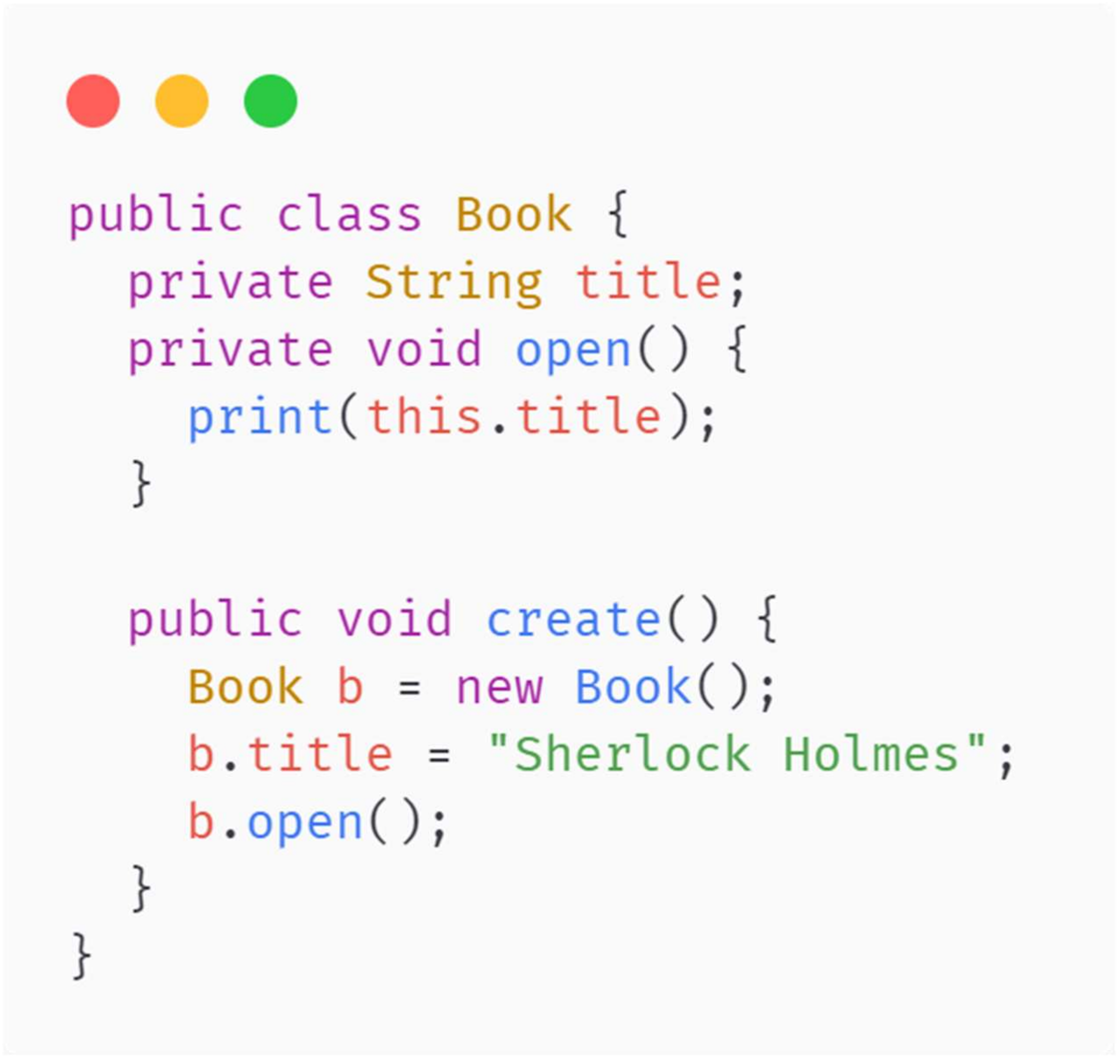


```java
public class Reader {
  public void read() {
    Book b = new Book();
    // the following code is wrong,
    // cannot access private members
    print(b.title);
  }
}
```

Private Modifier Note

If the object is declared in its own class, it can access its own private members.

Members: attributes and methods



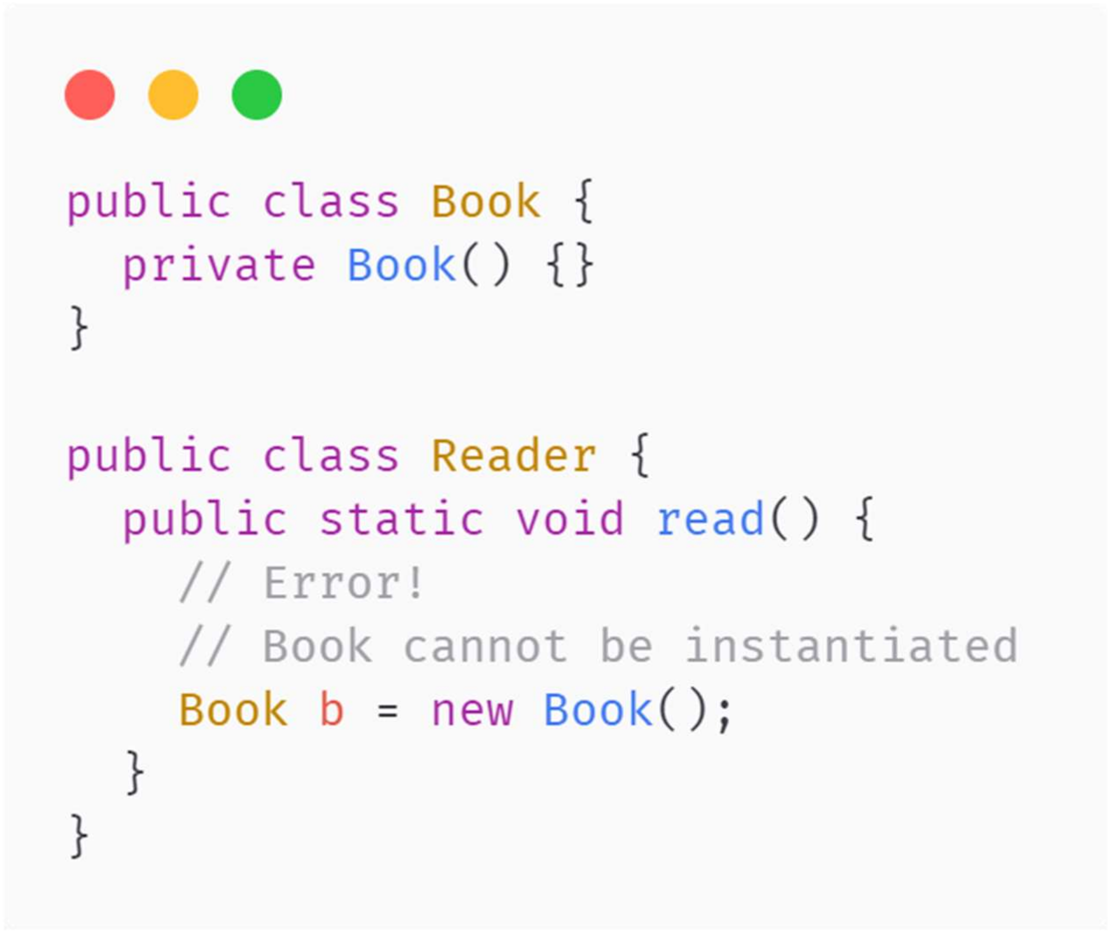
```
public class Book {
  private String title;
  private void open() {
    print(this.title);
  }

  public void create() {
    Book b = new Book();
    b.title = "Sherlock Holmes";
    b.open();
  }
}
```

Why Private Modifier?

- To protect data.
- To make code easy to maintain.
- Allow data to be set as read-only, write-only, or hide the data completely
    - Restricting access or hiding some data or behavior of an object is formally known as information hiding technique.

Private Constructor

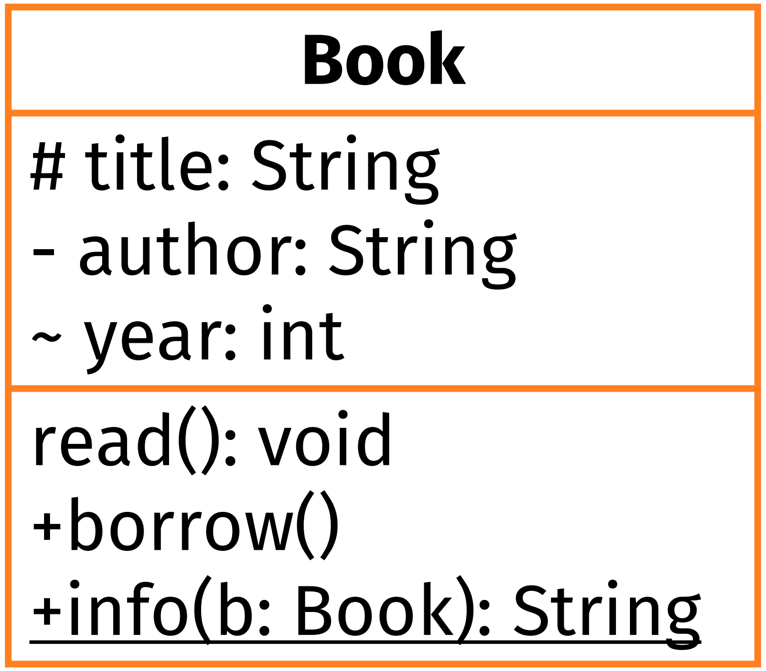
- Constructors can be set as private; however, classes with private constructor cannot be instantiated.



```
public class Book {
  private Book() {}
}

public class Reader {
  public static void read() {
    // Error!
    // Book cannot be instantiated
    Book b = new Book();
  }
}
```

Modifier in Class Diagram

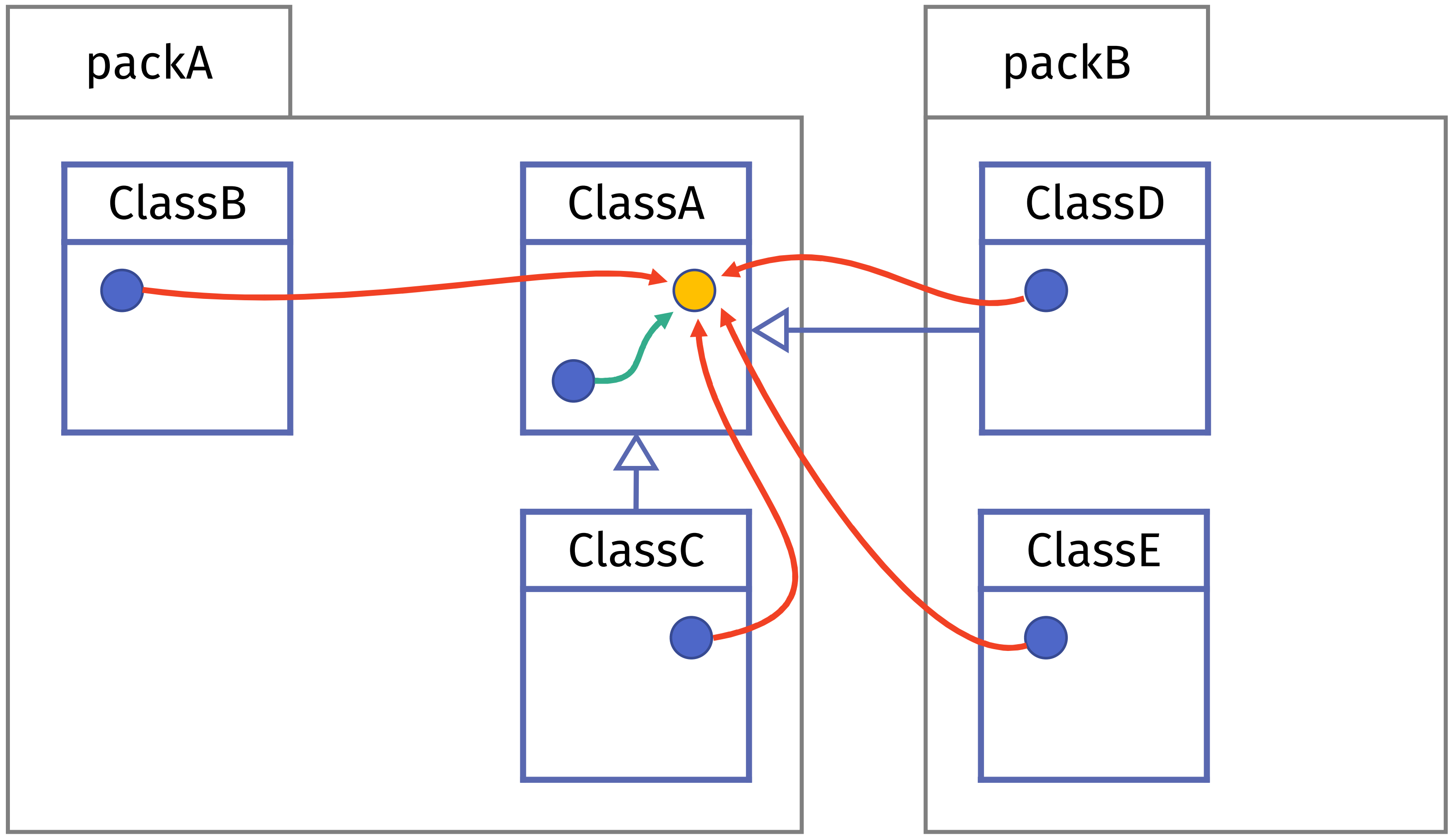
Modifier	Symbol
private	-
protected	#
default	~ (or blank)
public	+



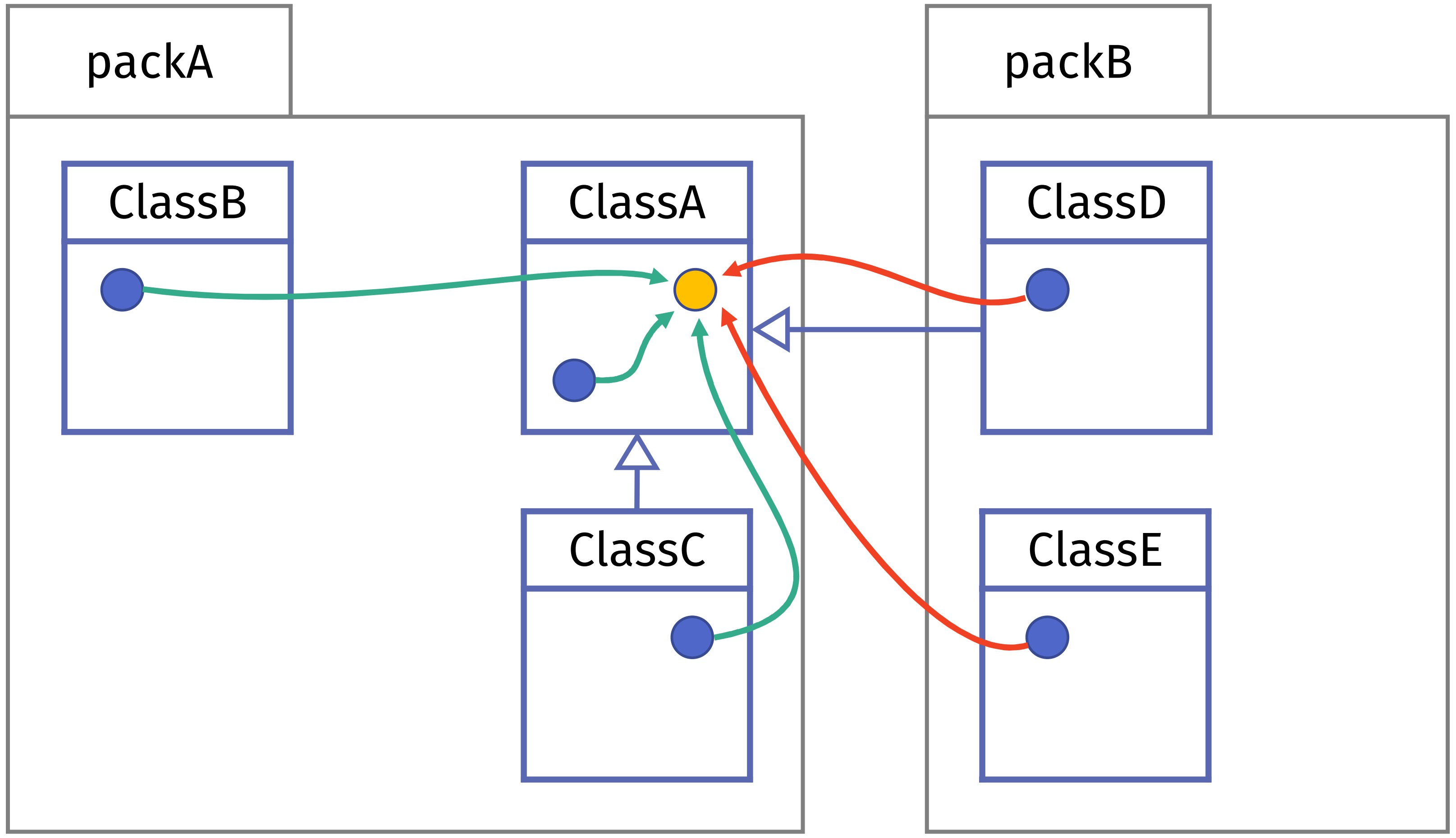
Access with Different Modifier

Access	private	default	protected	public
Same class	✓	✓	✓	✓
Same package subclass	✗	✓	✓	✓
Same package non-subclass	✗	✓	✓	✓
Different package subclass	✗	✗	✓	✓
Different package non-subclass	✗	✗	✗	✓

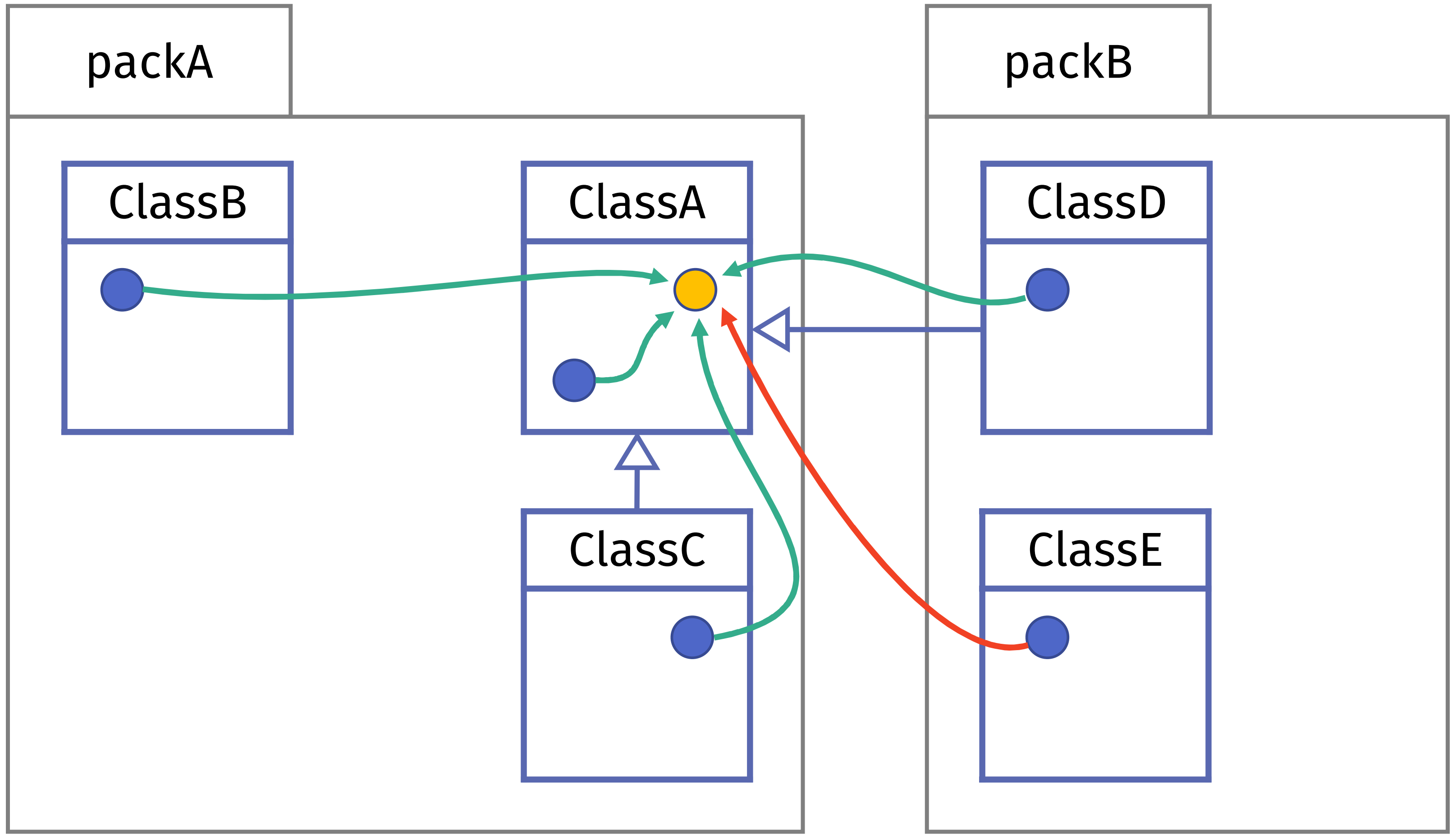
Member Access: Private



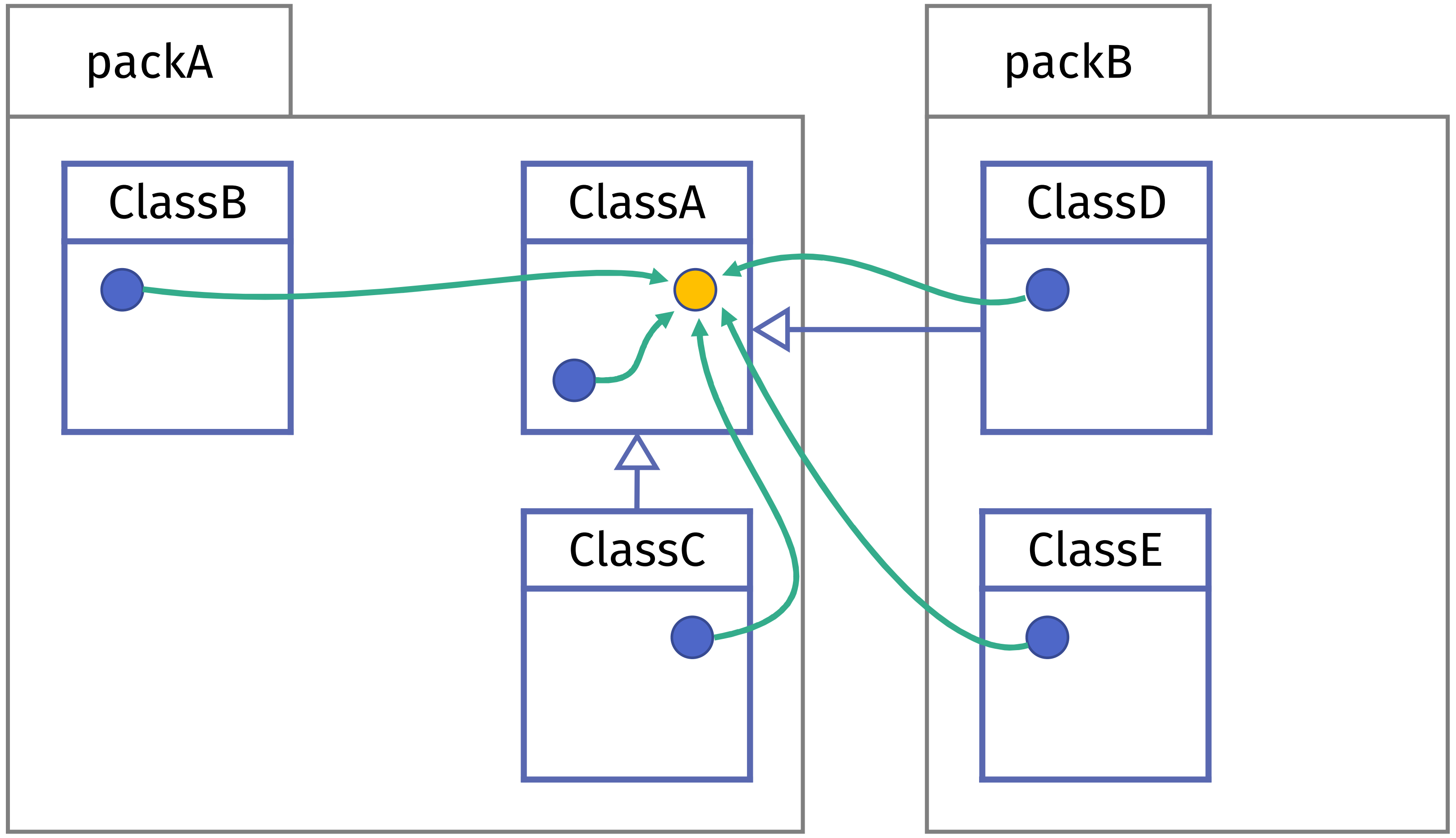
Member Access: Default



Member Access: Protected



Member Access: Public



Encapsulation

Encapsulation

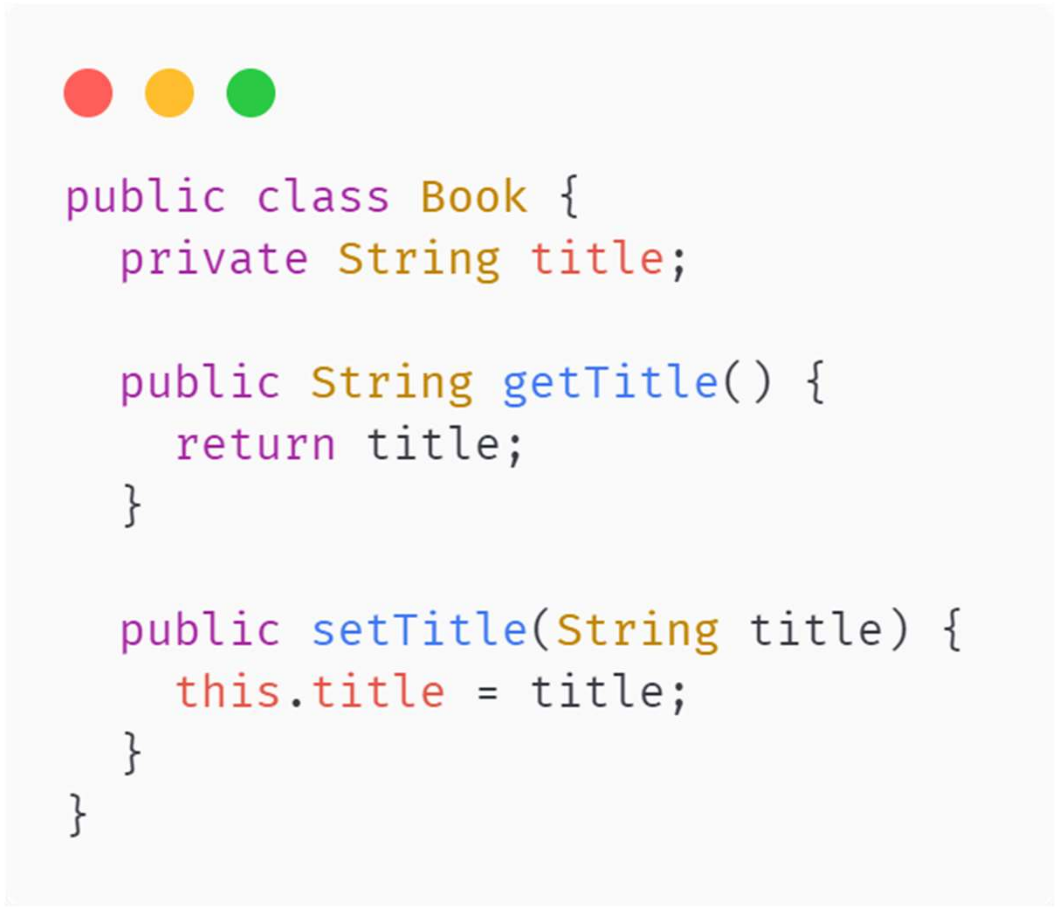
- Encapsulation in Java is a process of wrapping code and data together into a single unit.
  - Accessor/mutator methods of a private member
  - Wrapping/hide complex process into one simple accessible method

Accessor/Mutator Methods

The getter and setter methods to read and modify private properties.

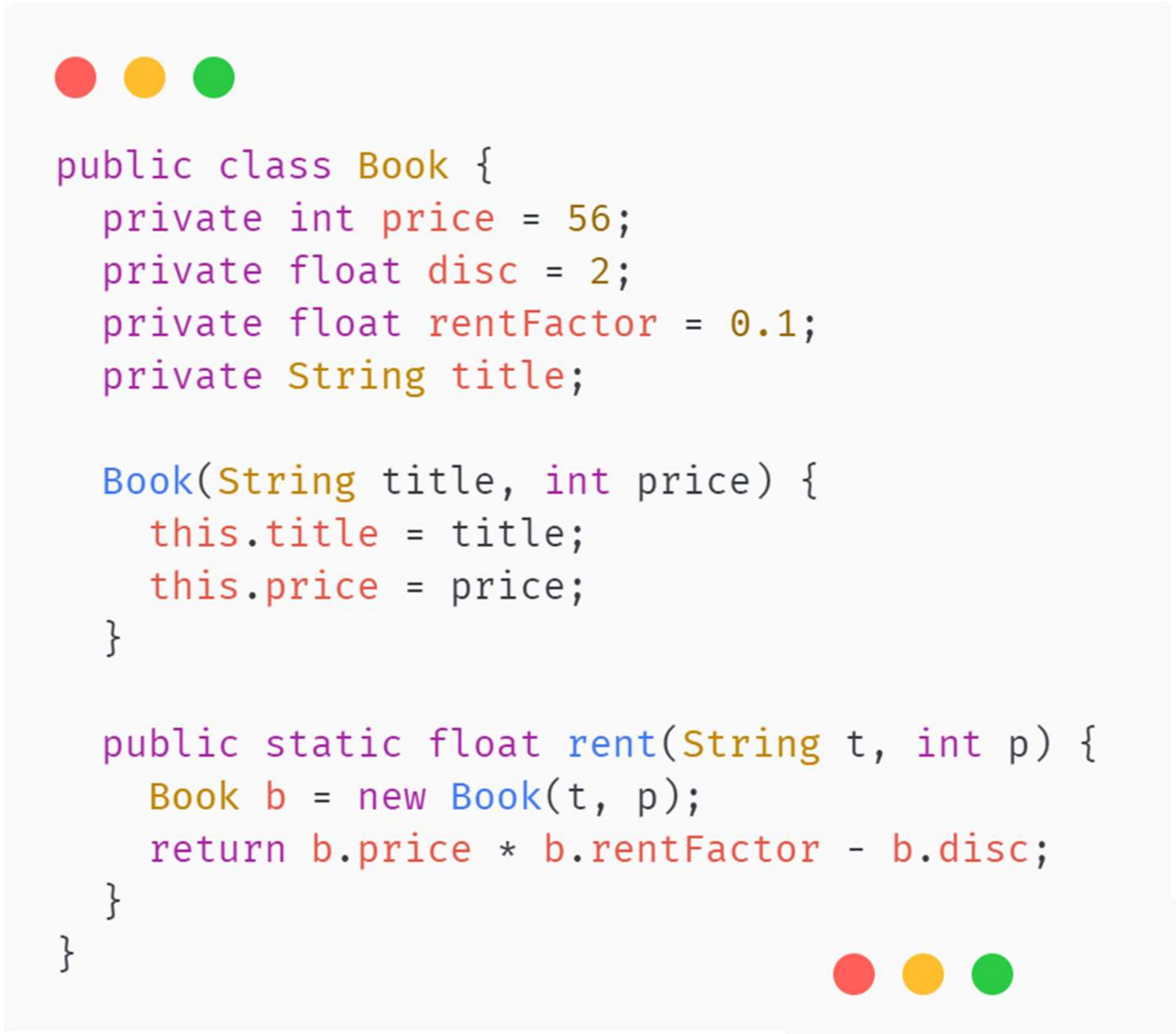
The title attribute is declared private, but it is encapsulated by its getter and setter method.

This way, its value can be accessed from outside of the Class using its encapsulating getter and setter methods.

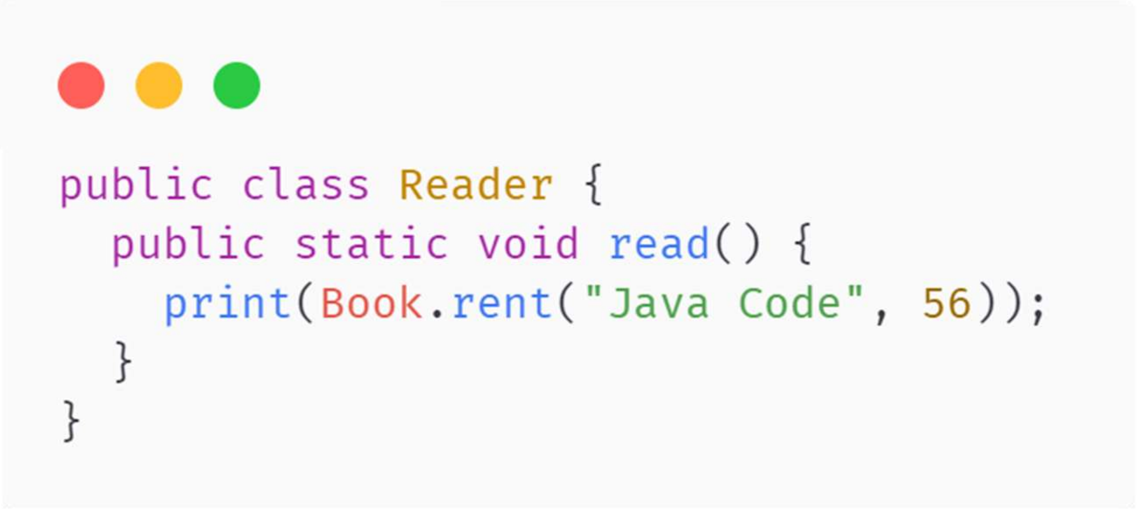


```java
public class Book {
  private String title;

  public String getTitle() {
    return title;
  }

  public setTitle(String title) {
    this.title = title;
  }
}
```

Wrapping Complex Process



```java
public class Book {
  private int price = 56;
  private float disc = 2;
  private float rentFactor = 0.1;
  private String title;

  Book(String title, int price) {
    this.title = title;
    this.price = price;
  }

  public static float rent(String t, int p) {
    Book b = new Book(t, p);
    return b.price * b.rentFactor - b.disc;
  }
}
```



```java
public class Reader {
  public static void read() {
    print(Book.rent("Java Code", 56));
  }
}
```

Class Relationship

Object-oriented Thinking

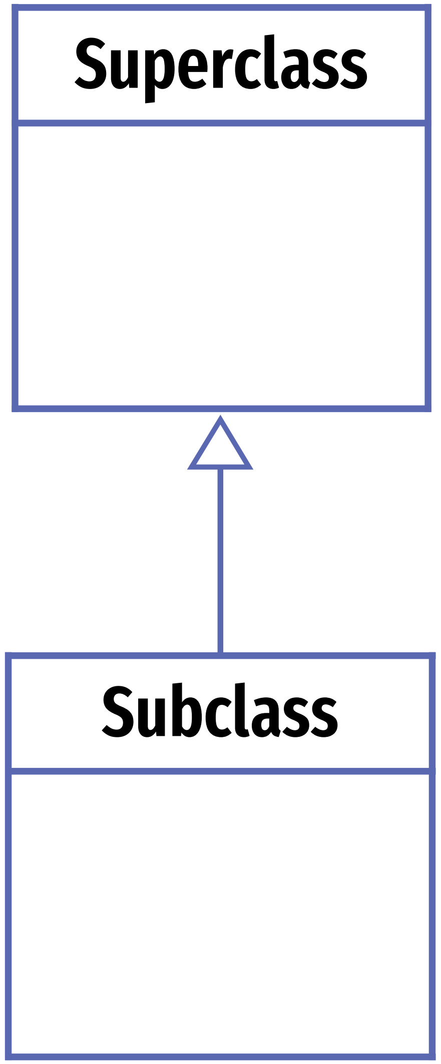
- Classes provide more flexibility and modularity for building reusable software.
- Object-oriented programming techniques benefits of developing reusable code using objects and classes.
- Classes may have relationship with other classes.

Class Relationship

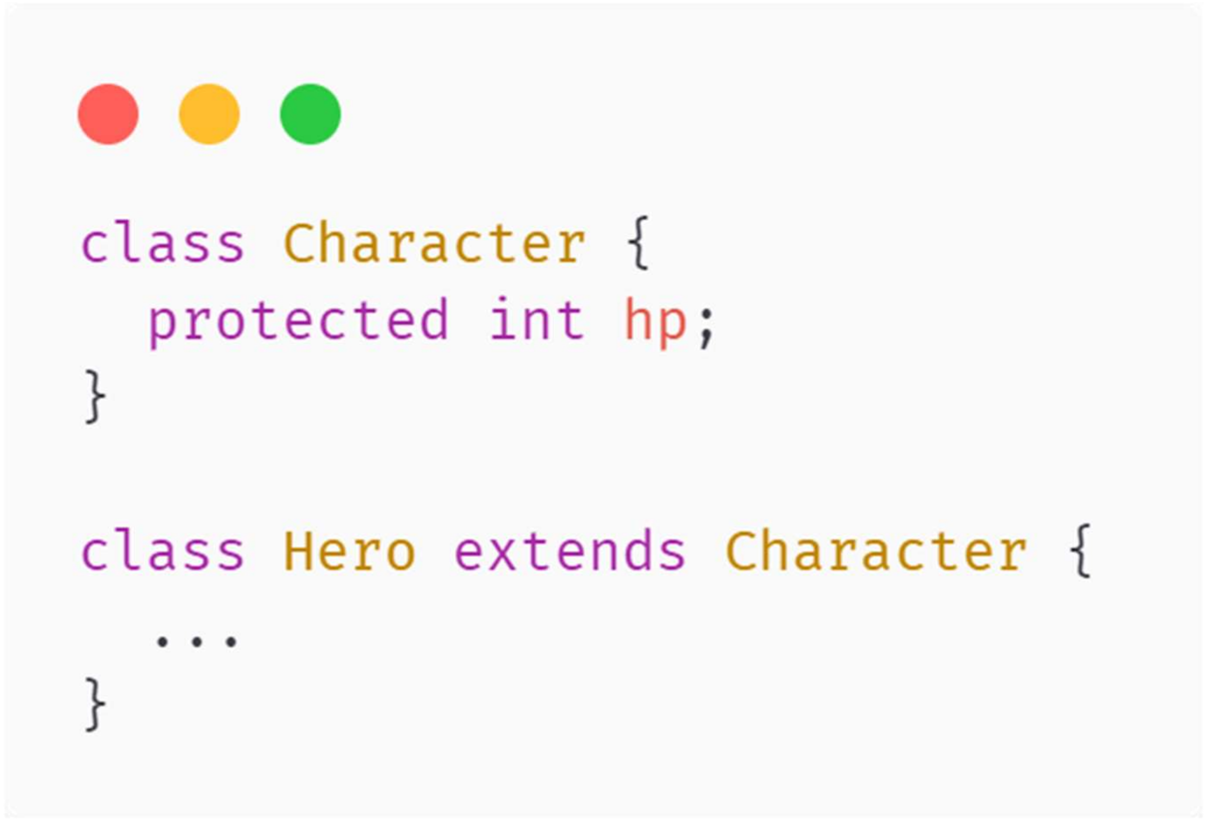
- Is-A, a generalization relationship representation when a class represent a more general form of another class
- Uses-A, a dependency relationship representation when a class uses an object of another class
- Has-A, an association relationship when an object of a class becomes a member of another class.
    - Aggregation and Composition

Generalization: "Is-A"

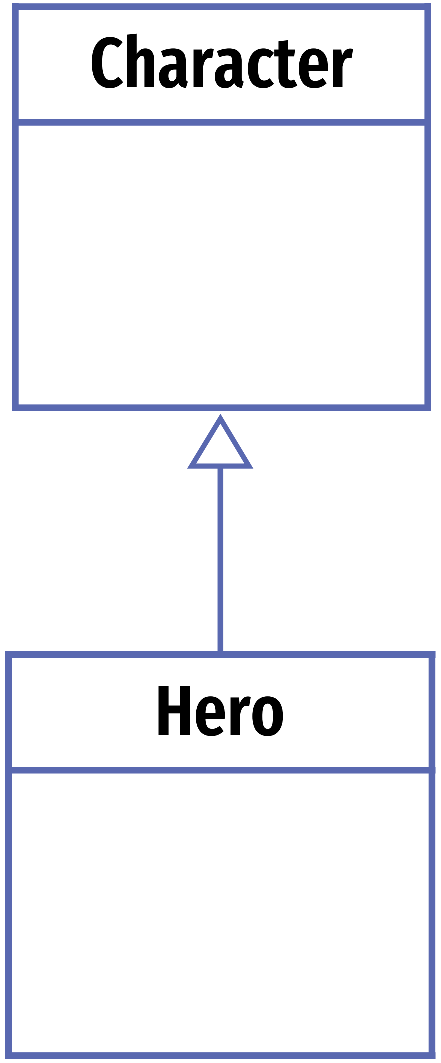
- Generalization establishes a relationship between a more general class (known as superclass) and a more specialized class (known as subclass).
- Is-A relationship defines the relationship between two classes in which one class extends another class



Generalization: "Is-A"

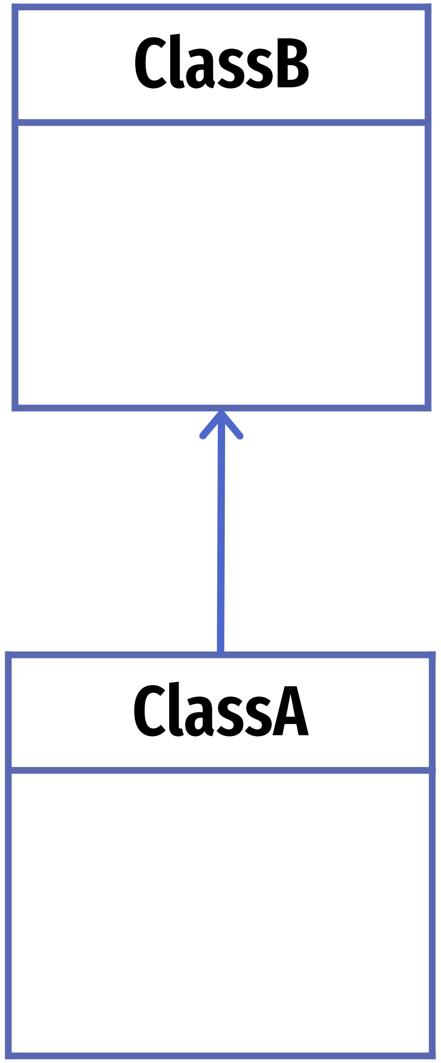


```
class Character {
  protected int hp;
}

class Hero extends Character {
  ...
}
```



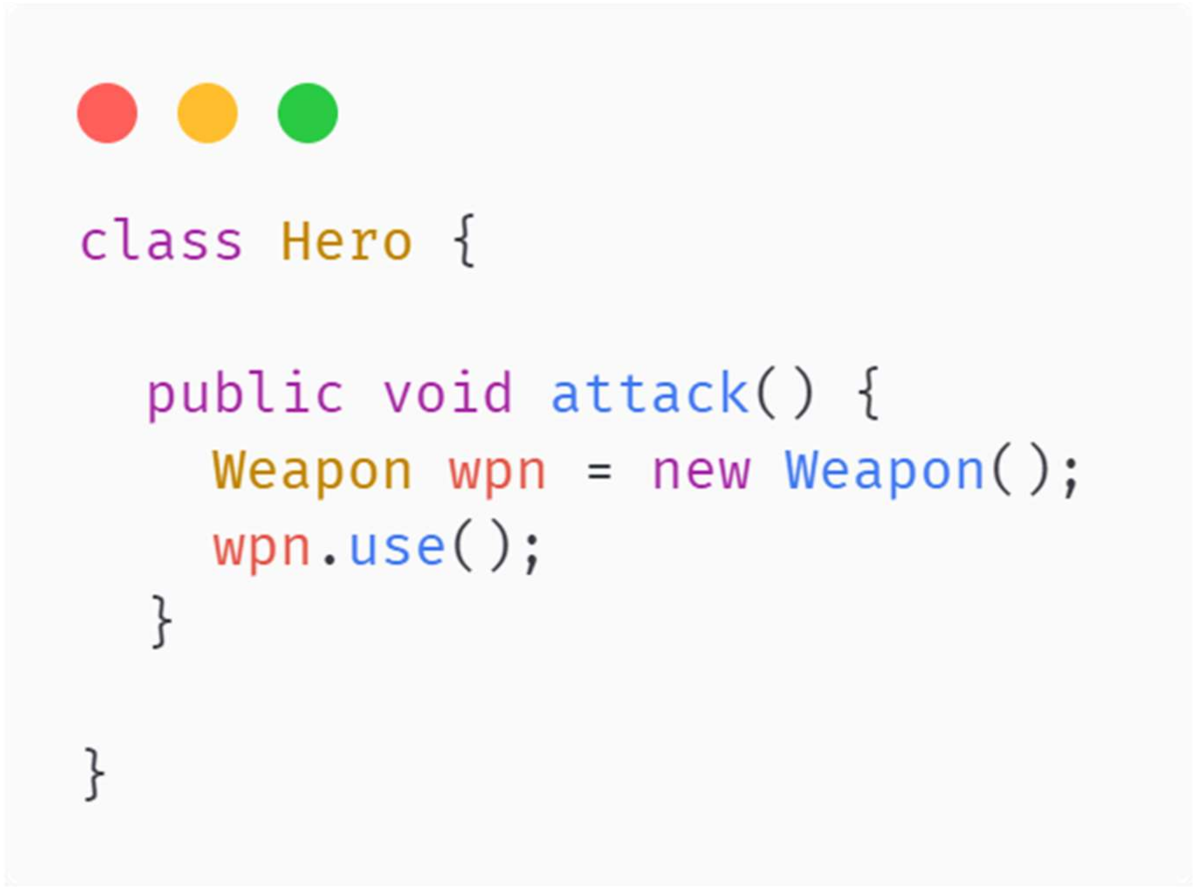
Dependency: "Uses-A"

- An object of other class is created inside a class's method.
- A method of a class uses an object of another class, it is called dependency in java.
- It is the most obvious and most general relationship in java.

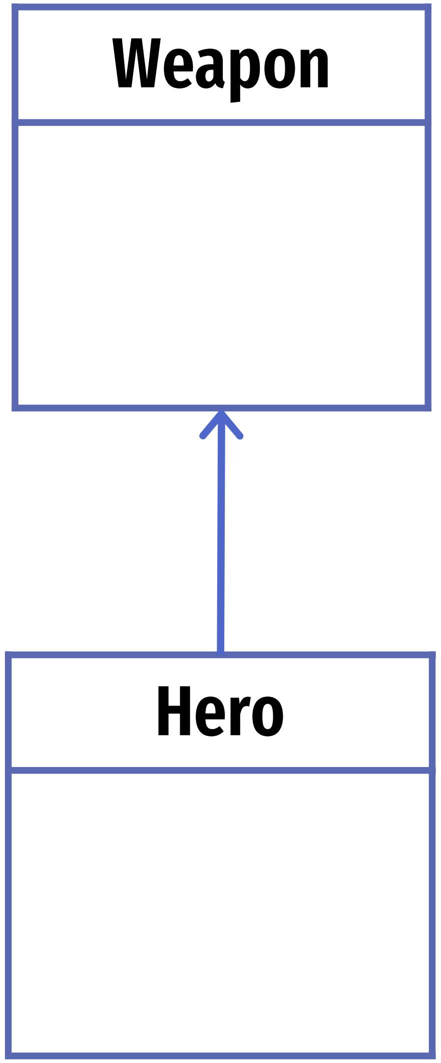


FILKOM

Dependence: "Uses-A"

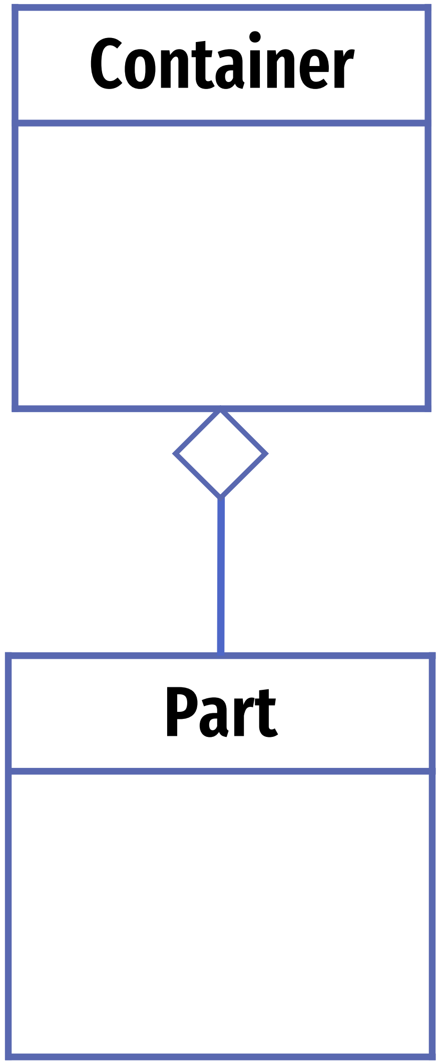


```
class Hero {

  public void attack() {
    Weapon wpn = new Weapon();
    wpn.use();
  }

}
```



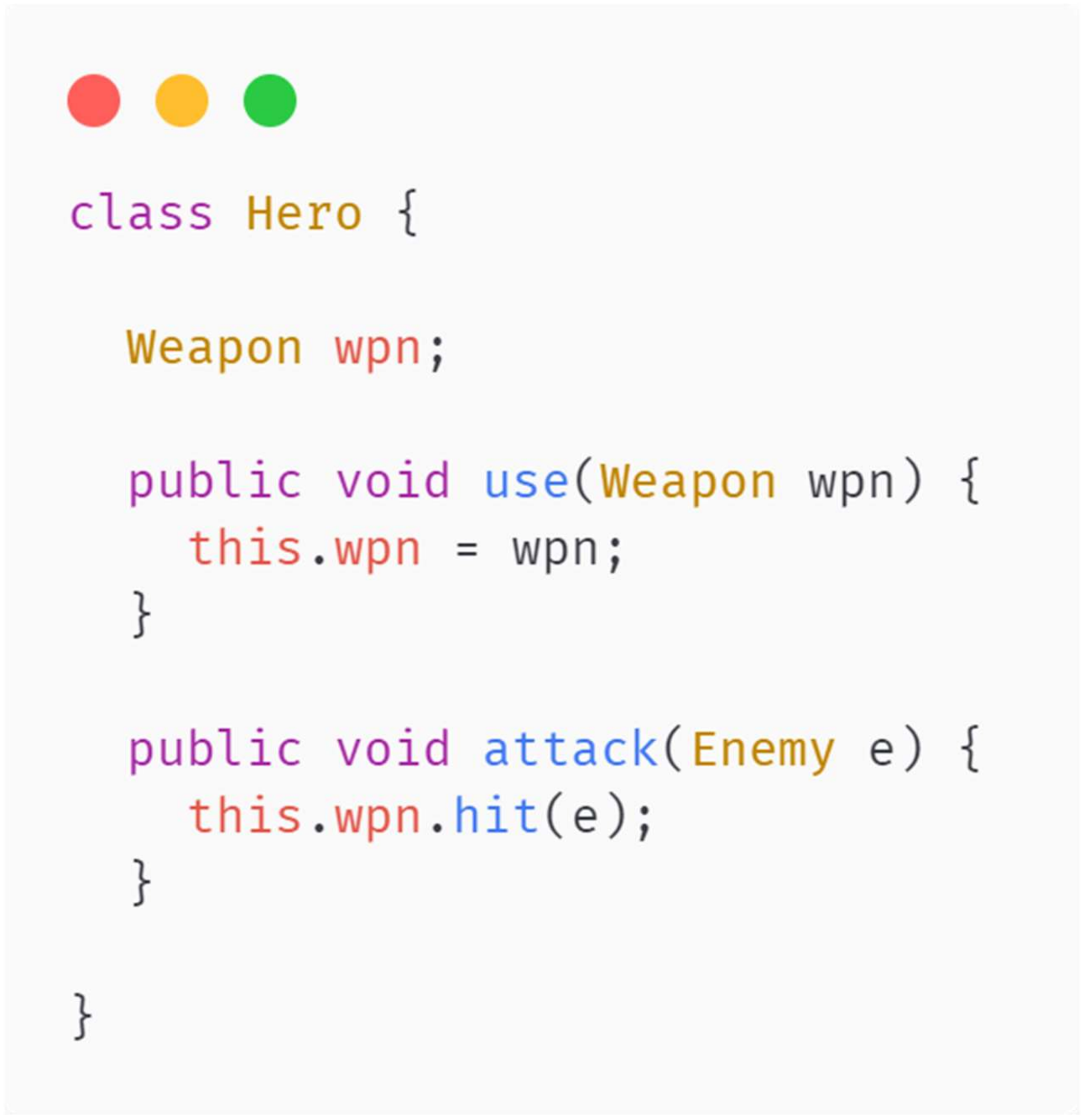
Association: "Has-A"

- Aggregation: An object of one class has members of object of another class.
- An object owns another object
- Both the container object and its member object may exists independently (conceptually).



Aggregation Example

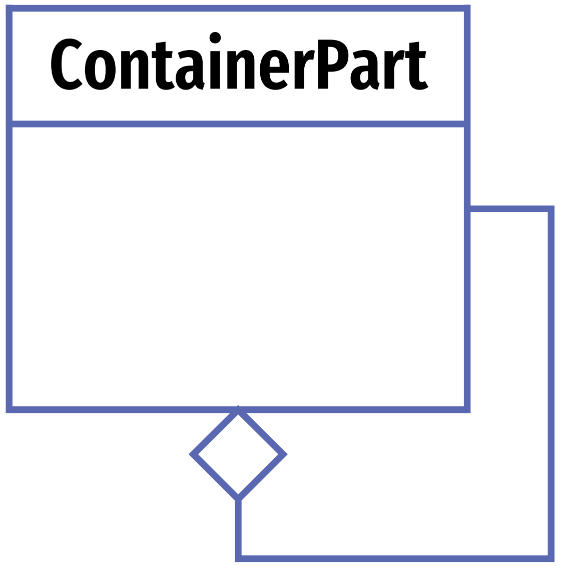
- A car and its engine
- A PC and its components, or…
- A Hero that uses a Weapon, which he own, to attack an Enemy



```java
class Hero {

  Weapon wpn;

  public void use(Weapon wpn) {
    this.wpn = wpn;
  }

  public void attack(Enemy e) {
    this.wpn.hit(e);
  }

}
```

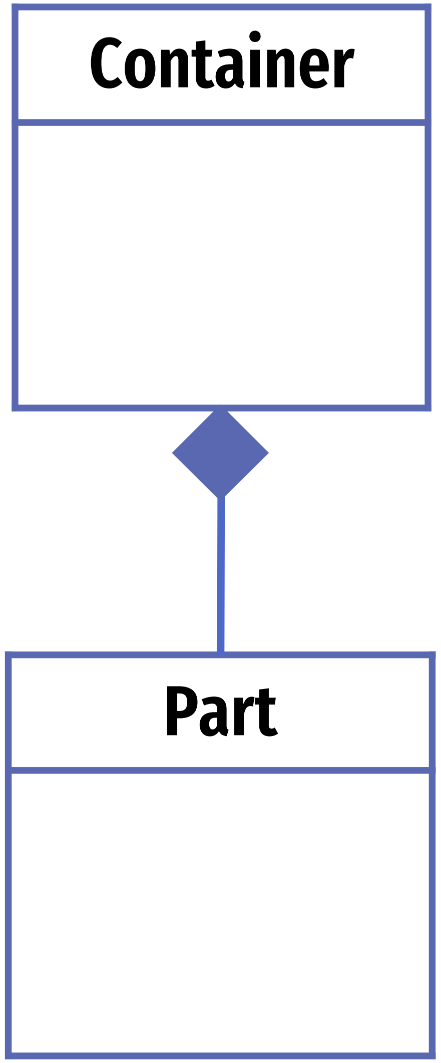
Self-Aggregation

- It is possible for a class to own an objects of its own class.
- Example:
  - Person and their Supervisor (which is also a Person)
  - A Product that contains another Product
  - A Person that married to another Person (Spouse).



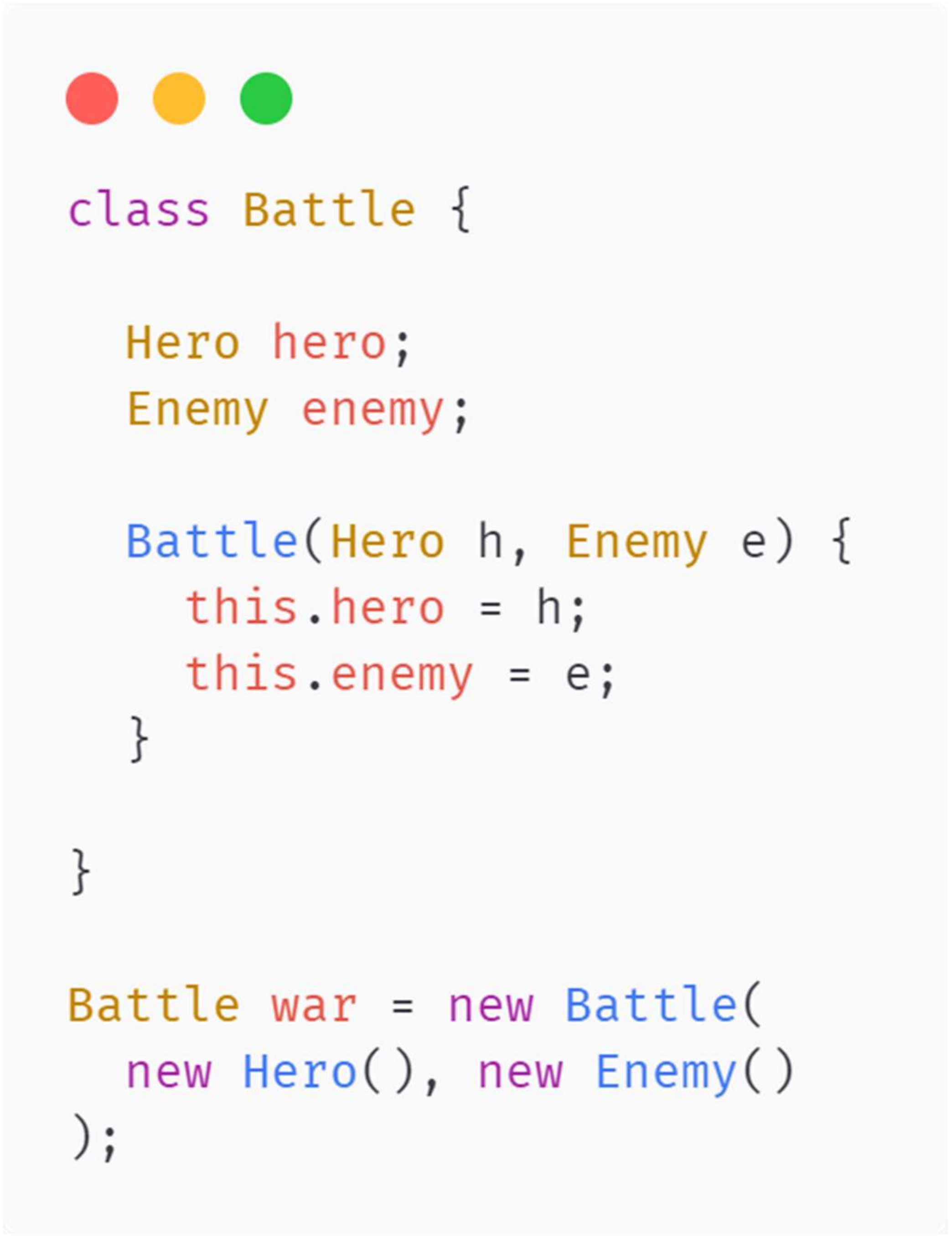
Association: "Has-A"

- Composition: An object of one class has members of object of another class.
- An object strongly owns another object
- Both the container object and its member object cannot exists independently (conceptually).
- Both objects must be exists together to make it senses.



Composition Example

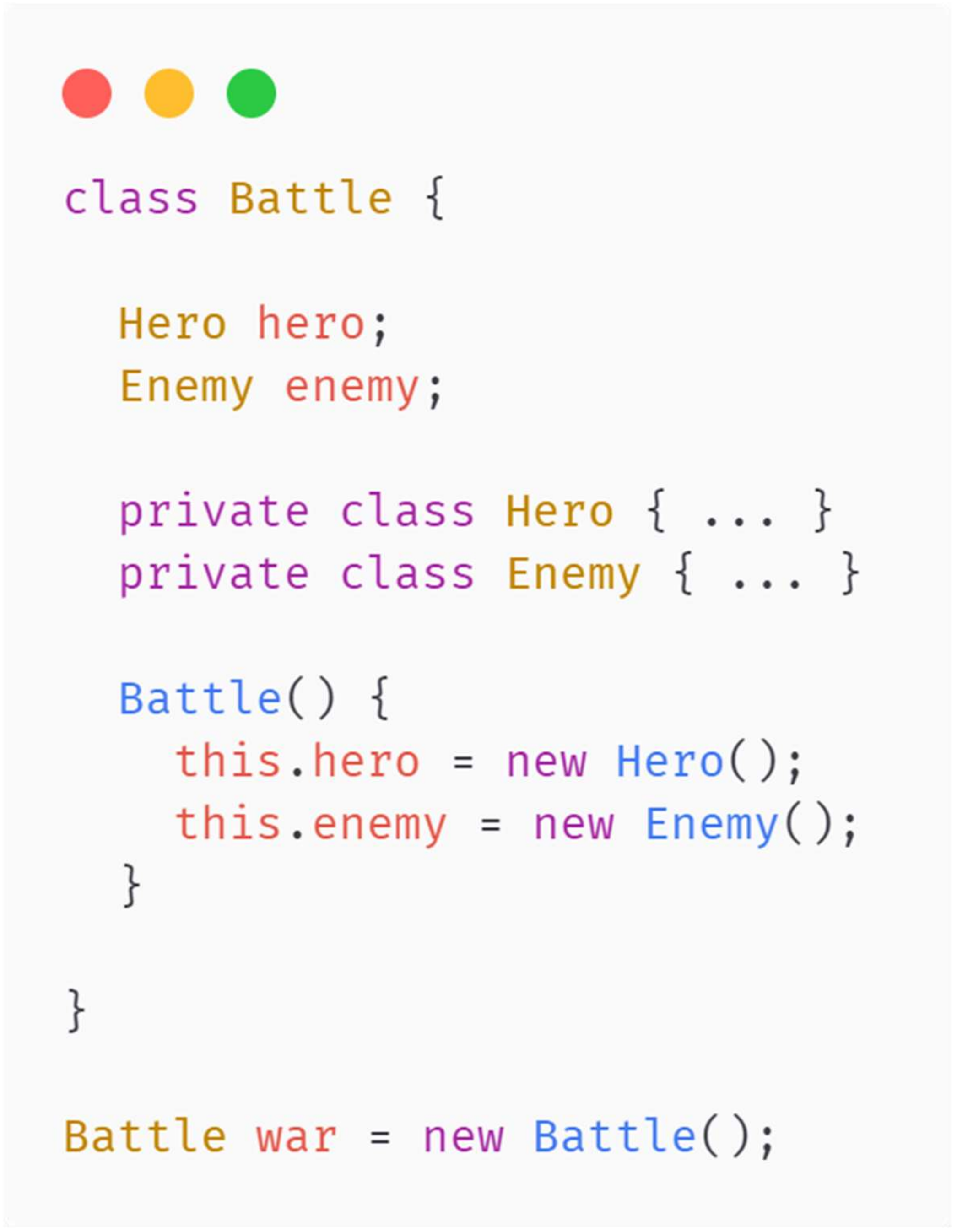
- Room and building
- Noodle and spaghetti
- Book and paper, or…
- Battle between a Hero and an Enemy



```java
class Battle {

  Hero hero;
  Enemy enemy;

  Battle(Hero h, Enemy e) {
    this.hero = h;
    this.enemy = e;
  }

}

Battle war = new Battle(
  new Hero(), new Enemy()
);
```

Composition Example (2)

- Battle between a Hero and an Enemy
- Implemented as private inner-classes of a container class



```
class Battle {

  Hero hero;
  Enemy enemy;

  private class Hero { ... }
  private class Enemy { ... }

  Battle() {
    this.hero = new Hero();
    this.enemy = new Enemy();
  }

}

Battle war = new Battle();
```

Questions?

Assignment

- Berdasarkan assignment sebelumnya, pada class Hero dan Enemy:
  - Ubah atribut hp, def, dan level menjadi read-only, dan buat method-method getter-nya.
  - Enkapsulasi method attack() ke dalam method doubleAttack() untuk menggandakan jumlah serangan Hero dan Enemy dalam satu kali kesempatan menyerang.

Assignment

- Pada class Hero dan Enemy:
    - Enkapsulasi method attack() dan heal() atau remedy() ke dalam satu method ultimate().
    - Dalam satu kali ultimate, attack dilakukan tiga kali jika hp Hero/Enemy kurang dari atau sama dengan 20% poin hp maksimalnya. Selain itu, attack yang dilakukan pada satu kali ultimate hanya dua kali.

Assignment

- Pada class Weapon:
  - Ubah seluruh atributnya menjadi read-only, dan buat method-method getter-nya.
  - Enkapsulasikan pada method use(), jika nilai condition kurang dari atau sama dengan 0, ubah nilai atribut isBroken menjadi true.
  - Enkapsulasikan pada method repair() untuk mengubah nilai condition menjadi 100 dan mengubah nilai isBroken menjadi false

Assignment

- Buat sebuah class Game yang mengagregasi Hero dan Enemy ke dalam sebuah game pertempuran.
- Buat hubungan antara Hero dan Weapon menjadi Dependency ("Uses-A")
- Buat hubungan antara Enemy dan Weapon menjadi Composition ("Has-A").
- Gambarkan diagram class-nya
- Implementasikan kode program Java-nya