Object Oriented Programming

By Watcharin Sarachai

Topic

- Access Modifiers
- Encapsulation

Access Modifiers

Access Modifiers

- You must have seen **public**, **private** and **protected** keywords while practicing java programs, these are called **access modifiers.**
- An access modifier restricts the access of a **class**, **constructor**, **data member** and **method** in another class.
- In java we have four access modifiers:

  1.default

  2.private

  3.protected

  4.public

1. Default access modifier

- When we do not mention any access modifier, it is called **default access modifier**.
- The scope of **default access modifier** is limited to the package only. This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside this package can access this class.
- Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package.

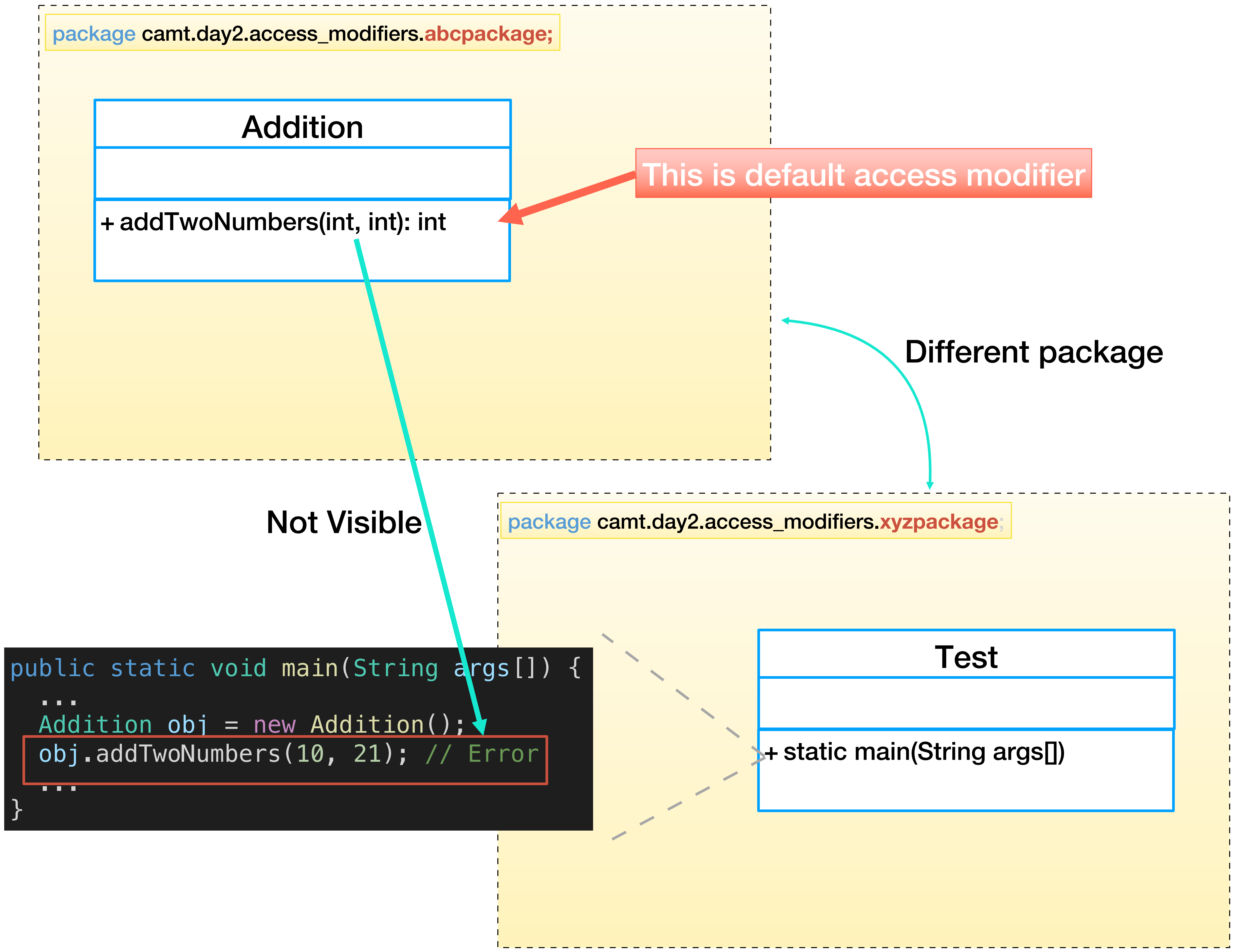
1. Default access modifier

- Default Access Modifier Example in Java
    - In this example we have two classes, **Test** class is trying to access the default method of **Addition** class.
    - Since class **Test** belongs to a different package, this program would throw compilation error, because the scope of default modifier is limited to the same package in which it is declared.

```java
package camt.day2.access_modifiers.abcpackage;


public class Addition {
  /*
   * Since we didn't mention any access modifier here, it would be considered as
   * default.
   */
  int addTwoNumbers(int a, int b) {
    return a + b;
  }
}
```

```java
package camt.day2.access_modifiers.xyzpackage;

/* We are importing the abcpackage
* but still we will get error because the
* class we are trying to use has default access
* modifier.
*/
import camt.day2.access_modifiers.abcpackage.*;

public class Test {
  public static void main(String args[]) {
    Addition obj = new Addition();
    /*
     * It will throw error because we are trying to access the default method in
     * another package
     */
    obj.addTwoNumbers(10, 21); // Error
  }
}
```

```
Output:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The method addTwoNumbers(int, int) from the type Addition is not visible
at xyzpackage.Test.main(Test.java:12)
```


package camt.day2.access_modifiers.abcpackage;
Addition
+ addTwoNumbers(int, int): int
This is default access modifier
Different package
Not Visible
package camt.day2.access_modifiers.xyzpackage;
Test
+ static main(String args[])

```
public static void main(String args[]) {
  ...
  Addition obj = new Addition();
  obj.addTwoNumbers(10, 21); // Error
  ...
}
```

2. Private access modifier

- The scope of private modifier is limited to the class only.

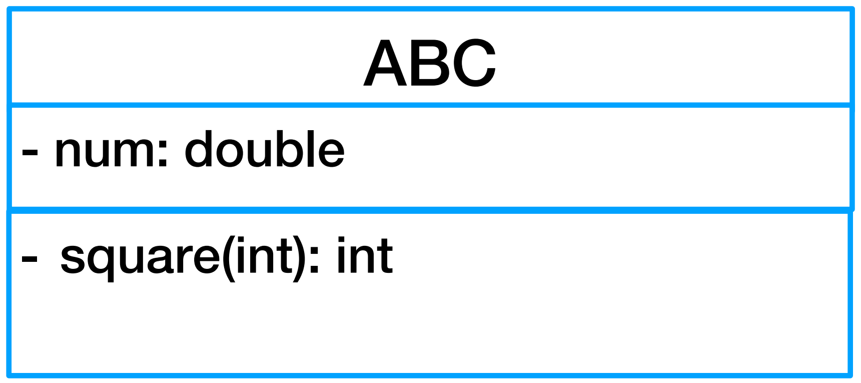
  1. Private Data members and methods are only accessible within the class

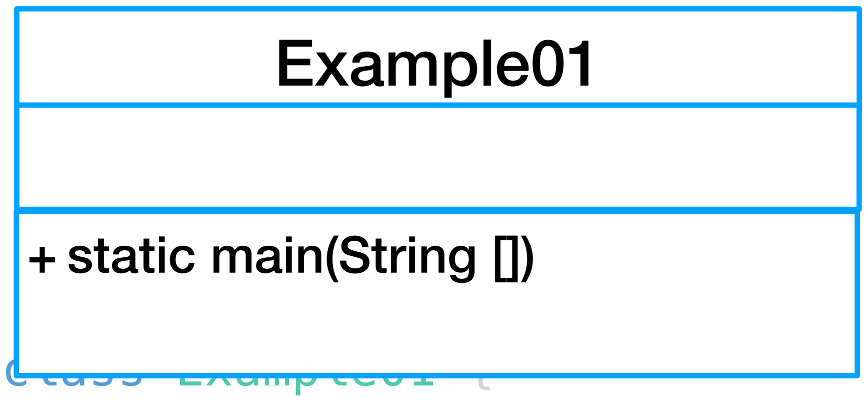
  2. **Class** and **Interface** cannot be declared as private

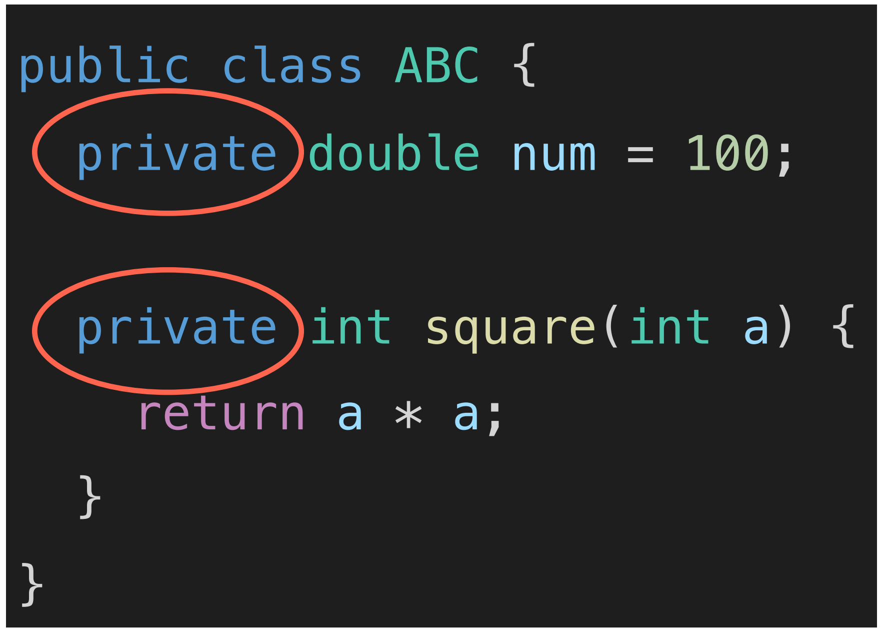
  3. If a class has private constructor then you cannot create the object of that class from outside of the class.

2. Private access modifier

- Private access modifier example in java
  - This example throws compilation error because we are trying to access the private data member and method of class ABC in the class Example.
  - The private data member and method are only accessible within the class.

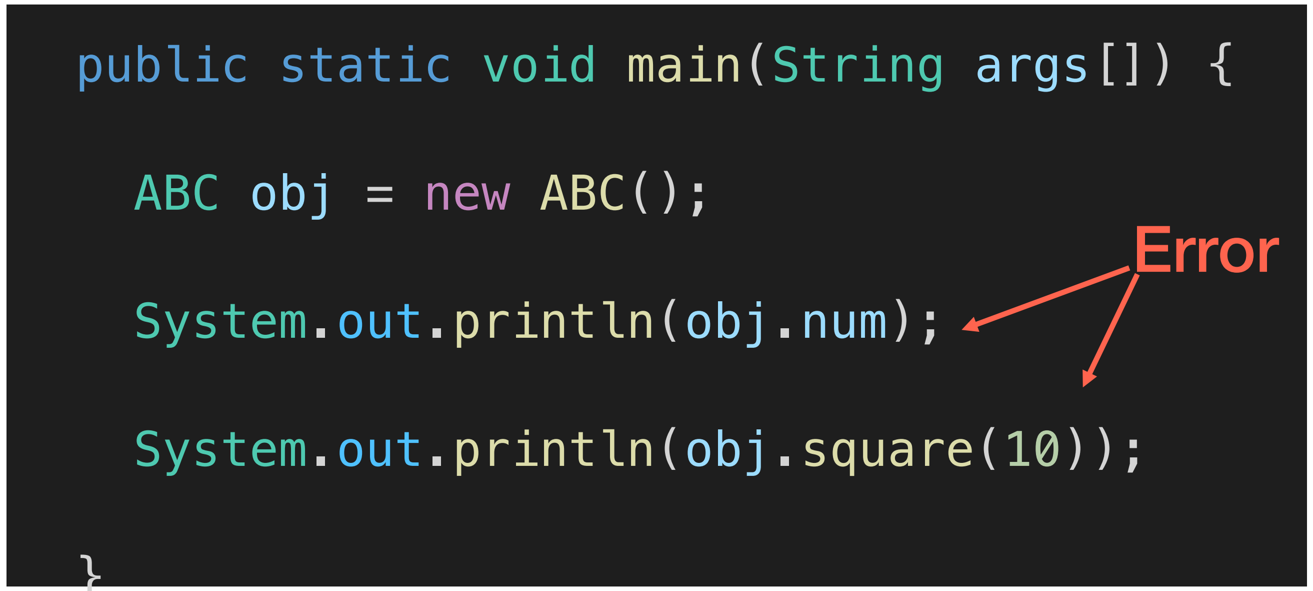






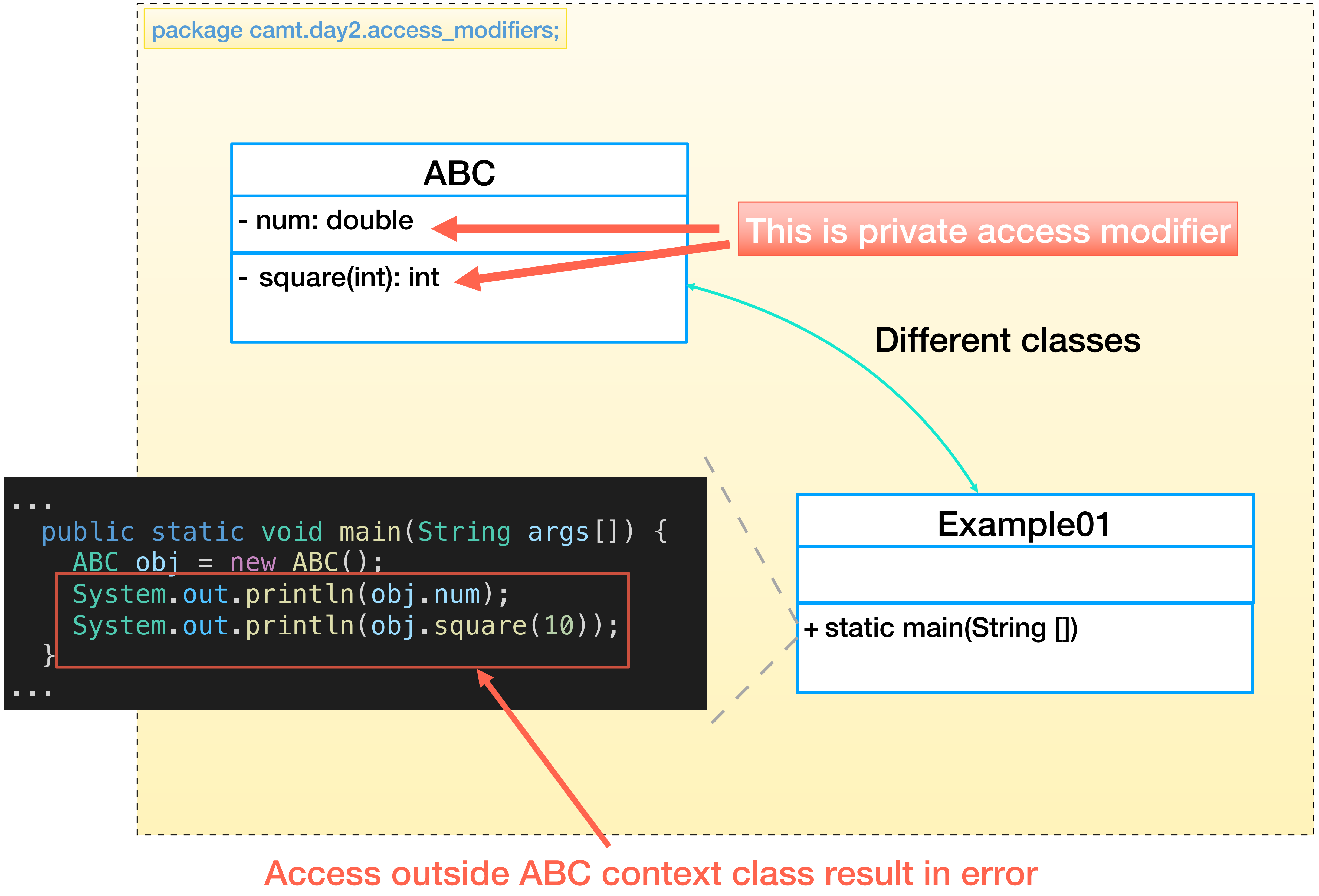
```java
public class ABC {
  private double num = 100;

  private int square(int a) {
    return a * a;
  }
}
```

public class Example01 {



```java
  public static void main(String args[]) {

    ABC obj = new ABC();

    System.out.println(obj.num);

    System.out.println(obj.square(10));

  }
```

}


package camt.day2.access_modifiers;
ABC
- num: double
- square(int): int
This is private access modifier
Different classes
Example01
+ static main(String [])
Access outside ABC context class result in error

```
...
  public static void main(String args[]) {
    ABC obj = new ABC();
    System.out.println(obj.num);
    System.out.println(obj.square(10));
  }
...
```

3. Protected Access Modifier

- Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package.

- We can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes.

- Classes cannot be declared protected. This access modifier is generally used in a parent child relationship.

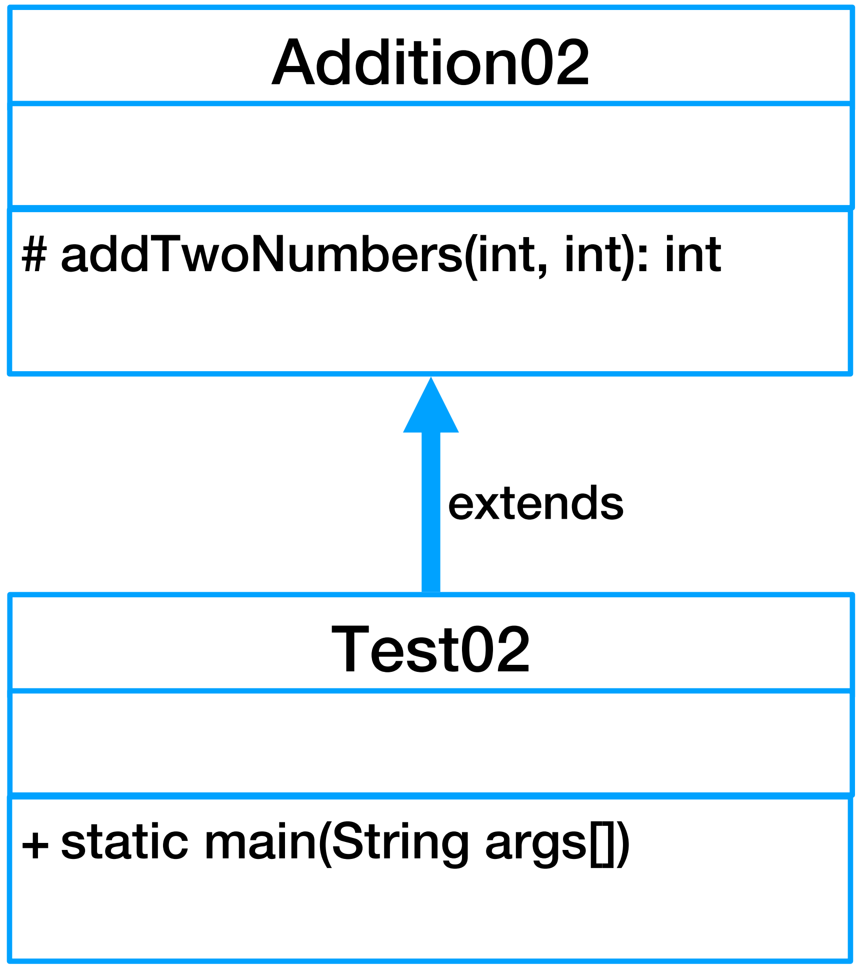
3. Protected Access Modifier

- Protected access modifier example in Java
    - In this example the class **Test** which is present in another package is able to call the addTwoNumbers() method, which is declared **protected**.
    - This is because the **Test** class **extends** class **Addition** and the **protected** modifier allows the access of **protected members** in **subclasses** (in any packages).

3. Protected Access Modifier

```
package camt.day2.access_modifiers.abcpackage;

public class Addition02 {
  protected int addTwoNumbers(int a, int b) {
    return a + b;
  }
}
```

```
package camt.day2.access_modifiers.xyzpackage;

import camt.day2.access_modifiers.abcpackage.*;

public class Test02 extends Addition02 {
  public static void main(String args[]) {
    Test02 obj = new Test02();
    obj.addTwoNumbers(10, 21);
  }
}
```

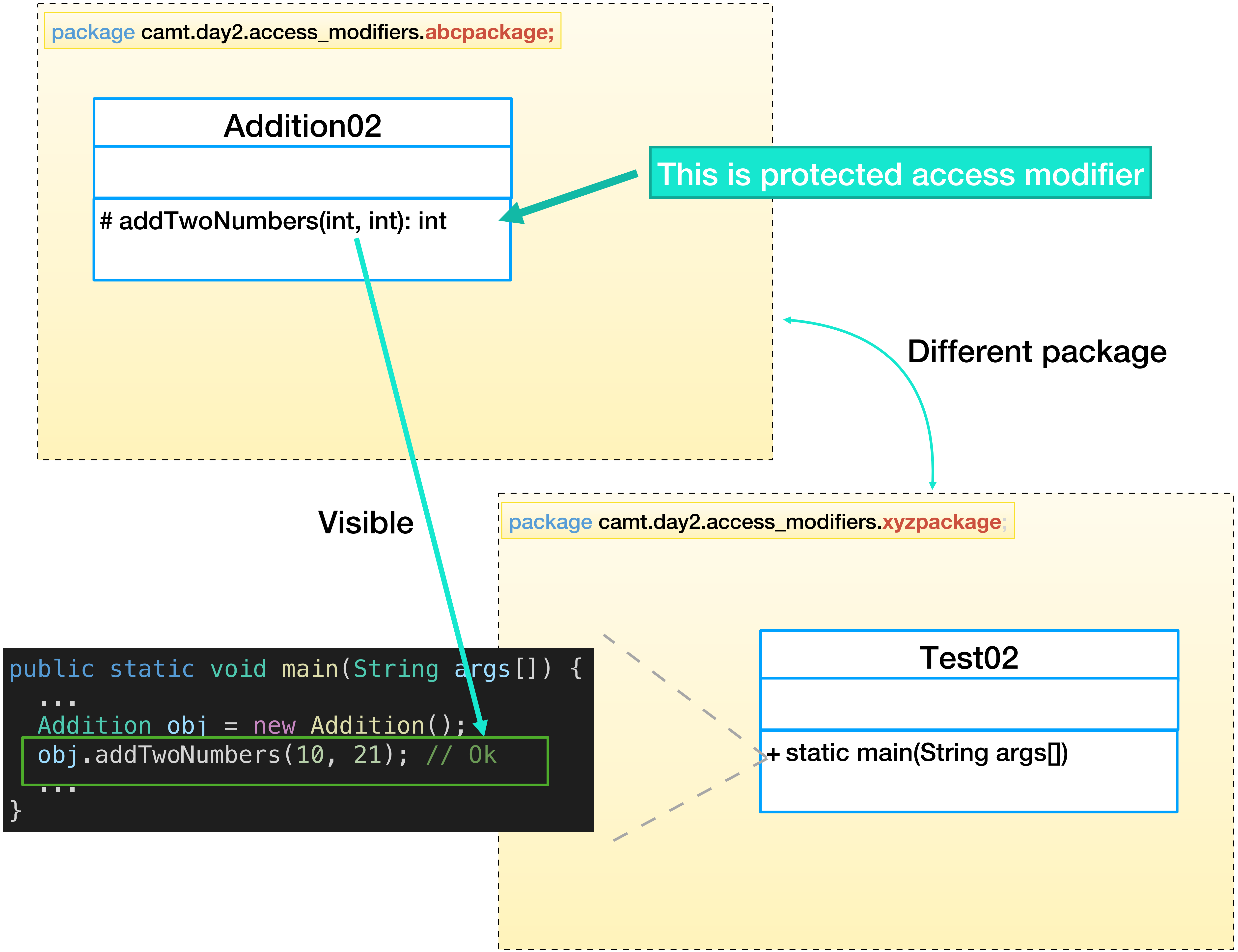


3. Protected Access Modifier

```java
package camt.day2.access_modifiers.abcpackage;

public class Addition02 {
  protected int addTwoNumbers(int a, int b) {
    return a + b;
  }
}
```

```java
package camt.day2.access_modifiers.xyzpackage;

import camt.day2.access_modifiers.abcpackage.*;

public class Test02 extends Addition02 {
  public static void main(String args[]) {
    Addition02 obj = new Addition02();
    obj.addTwoNumbers(10, 21);
  }
}
```

Error ??


package camt.day2.access_modifiers.abcpackage;
Addition02
# addTwoNumbers(int, int): int
This is protected access modifier
Different package
Visible
package camt.day2.access_modifiers.xyzpackage;
Test02
+ static main(String args[])

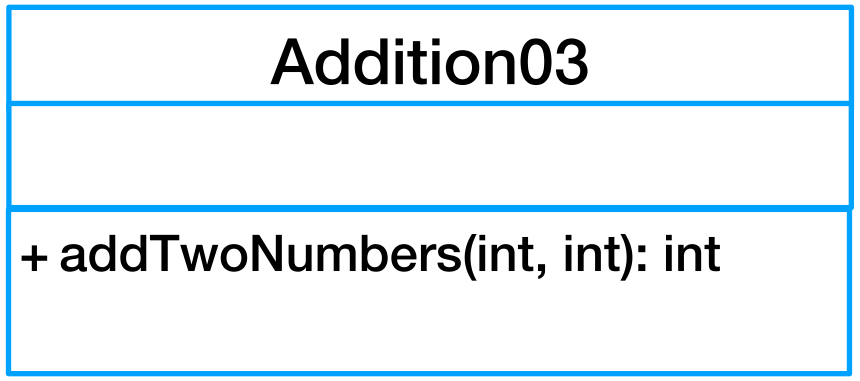
```
public static void main(String args[]) {
  ...
  Addition obj = new Addition();
  obj.addTwoNumbers(10, 21); // Ok
  ...
}
```

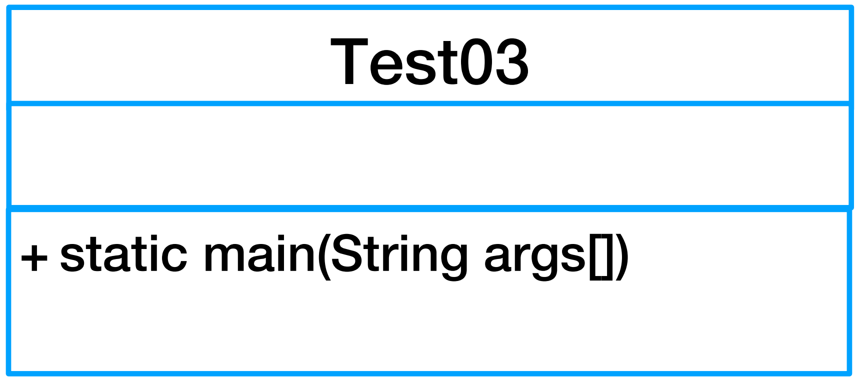
4. Public access modifier

- The members, methods and classes that are declared **public** can be accessed from **anywhere**.
- This modifier doesn't put any restriction on the access.
- public access modifier example in java
  - Lets take the same example that we have seen in previous slide but this time the method **addTwoNumbers()** has **public** modifier and class **Test** is able to access this method without even extending the **Addition** class.
  - This is because **public** modifier has visibility everywhere.

4. Public access modifier

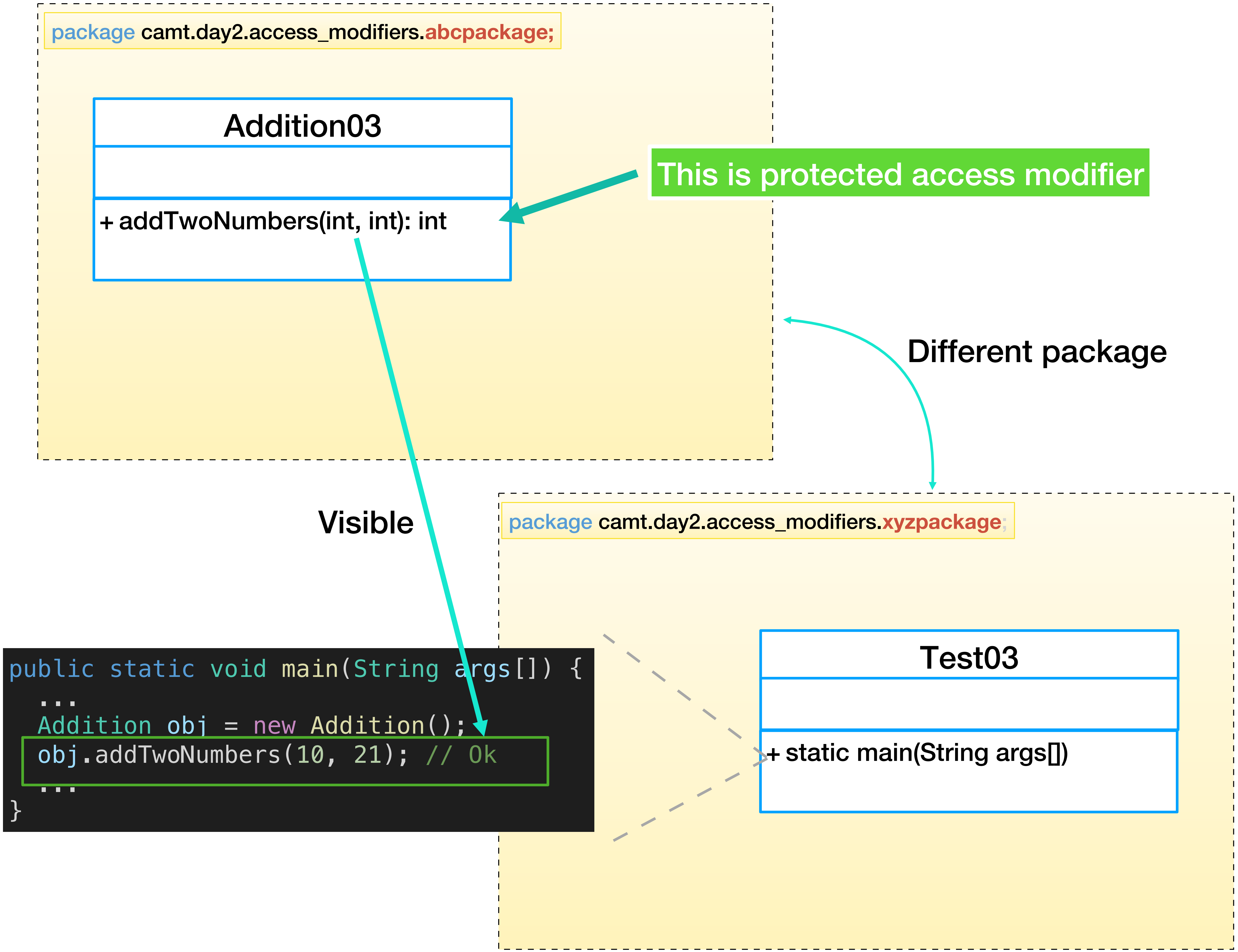
```java
package camt.day2.access_modifiers.abcpackage;

public class Addition03 {
  public int addTwoNumbers(int a, int b) {
    return a + b;
  }
}
```





```java
package camt.day2.access_modifiers.xyzpackage;

import camt.day2.access_modifiers.abcpackage.*;

public class Test03 {
  public static void main(String args[]) {
    Addition03 obj = new Addition03();
    obj.addTwoNumbers(10, 21);
  }
}
```

No error


package camt.day2.access_modifiers.abcpackage;
Addition03
+ addTwoNumbers(int, int): int
This is protected access modifier
Different package
Visible
package camt.day2.access_modifiers.xyzpackage;
Test03
+ static main(String args[])

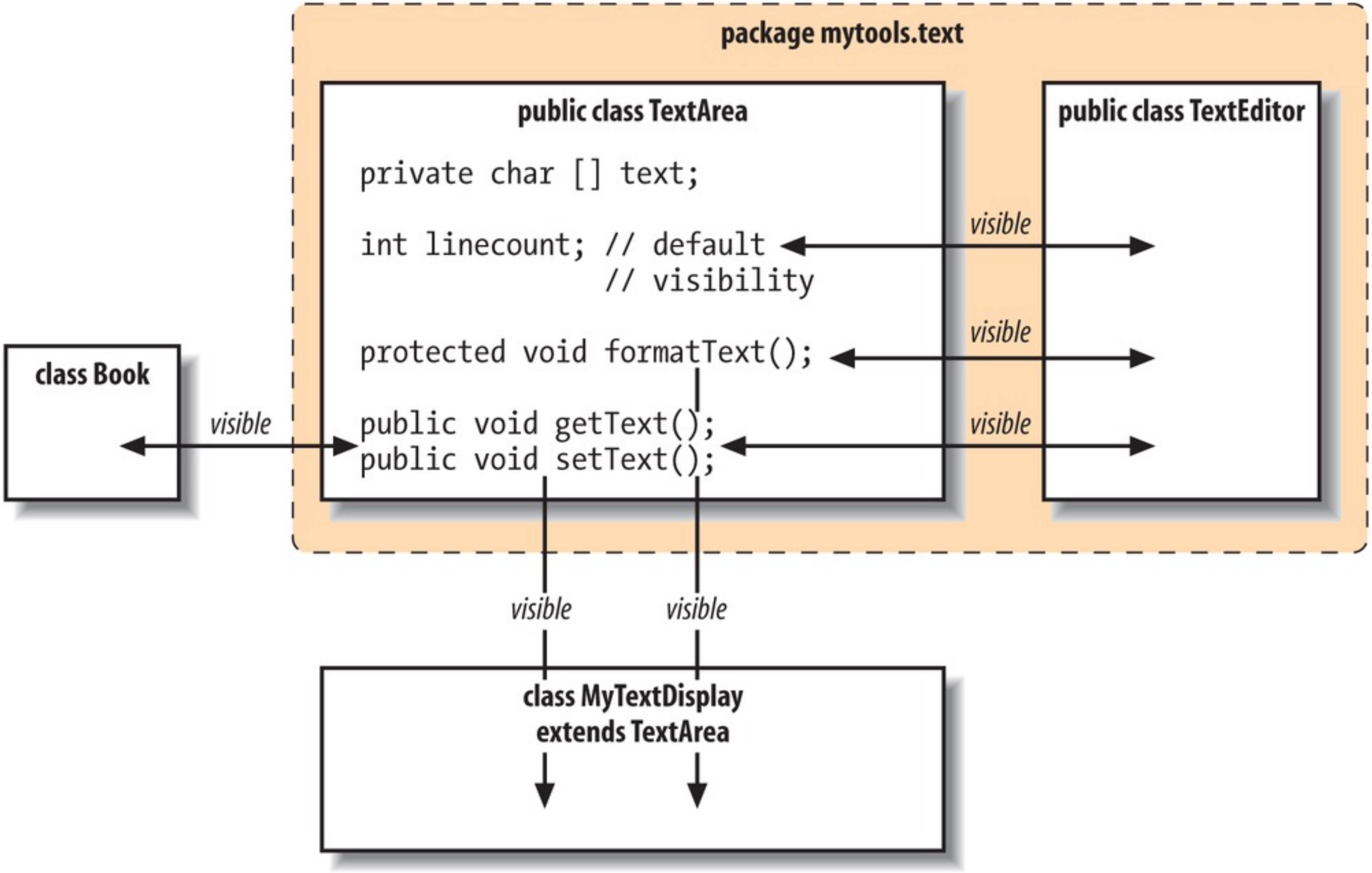
```
public static void main(String args[]) {
  ...
  Addition obj = new Addition();
  obj.addTwoNumbers(10, 21); // Ok
  ...
}
```

Access Modifiers in Java

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Access Modifiers in Java

	private	**default**	**protected**	**public**
Class	No	Yes	No	Yes
Nested Class	Yes	Yes	Yes	Yes
Constructor	Yes	Yes	Yes	Yes
Method	Yes	Yes	Yes	Yes
Field	Yes	Yes	Yes	Yes


package mytools.text
public class TextArea
private char [] text;
int linecount; // default
// visibility
protected void formatText();
public void getText();
public void setText();
public class TextEditor
visible
visible
visible
class Book
visible
visible
visible
class MyTextDisplay
extends TextArea

Do It Yourself

- Fix error in the class ClockReader and SmartClock in package ***camt.day2.access_modifiers.example***

Encapsulation

Encapsulation

Encapsulation binds data and its related methods together within a class. It also protects the data by making fields private and giving access to them only through their related methods.

Goals

Data Protection

Code Readability

In Java

Private Fields and Public Getter and Setter Methods.

Syntax:

`private` `getData()` `setData()`

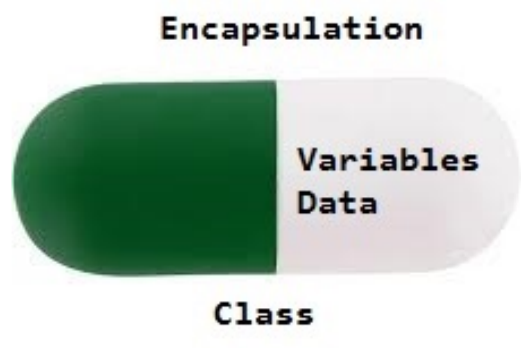
Encapsulation

- Encapsulation allows us to protect the data stored in a class from system-wide access.

- Encapsulation is safeguarded the internal contents of a class like a real-life capsule.

- We can implement encapsulation in Java by keeping the fields (*class variables*) private and providing public getter and setter methods to each of them.

- If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.



Encapsulation

- **Encapsulation in Java:**
    - Restricts direct access to data members (*fields*) of a **class**.
    - Fields are set to **private**.
    - Each field has a **getter** and **setter** method
    - **Getter** methods return the field
    - **Setter** methods let us change the value of the field
- The public **getter** and **setter** methods are used to update and read the **private** data fields then the outside class can access those private data fields via public methods.

Get and Set

- We learned from the previous that private variables can only be accessed within the same class (*an outside class has no access to it*). However, it is possible to access them if we provide public get and set methods.

- The get method returns the variable value, and the set method sets the value.

- Syntax for both is that they start with either get or set, followed by the name of the variable, with the first letter in **upper case**:

```java
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

Get and Set

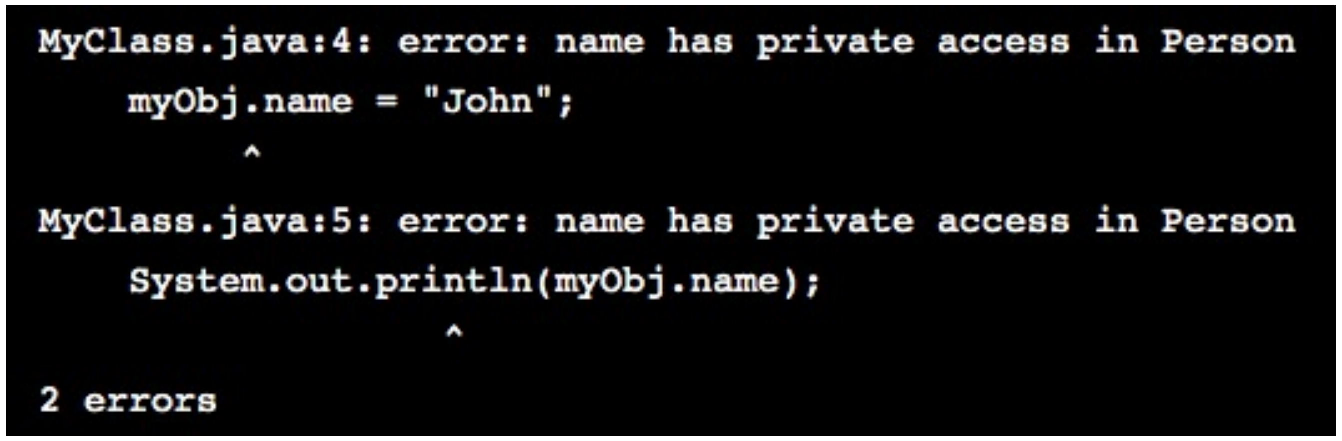
- Example explained
  - The get method returns the value of the variable name.
  - The set method takes a parameter (newName) and assigns it to the name variable. The this keyword is used to refer to the current object.
  - However, as the name variable is declared as private, we cannot access it from outside this class:

```java
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

```java
public class MyClass {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.name = "John";  // error
    System.out.println(myObj.name); // error
  }
}
```

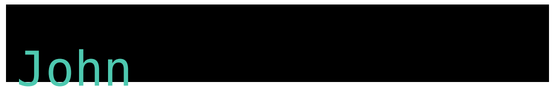
Get and Set

- However, as we try to access a private variable, we get an error:

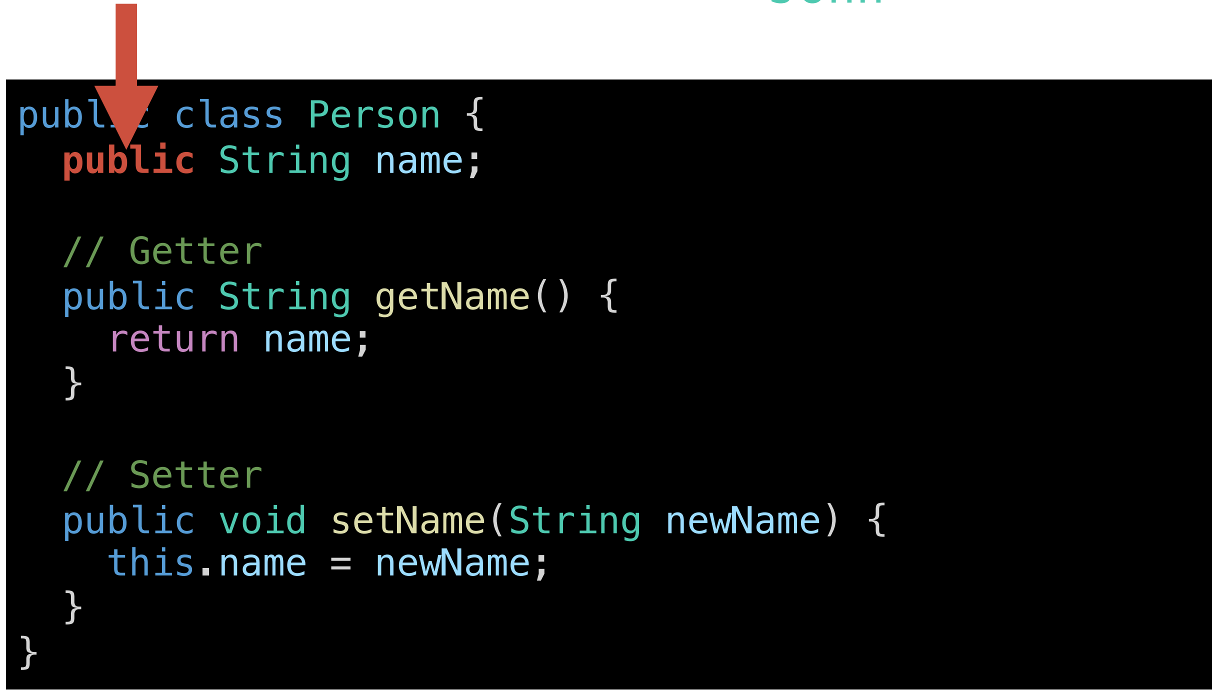


```
MyClass.java:4: error: name has private access in Person
    myObj.name = "John";
         ^
MyClass.java:5: error: name has private access in Person
    System.out.println(myObj.name);
                            ^
2 errors
```

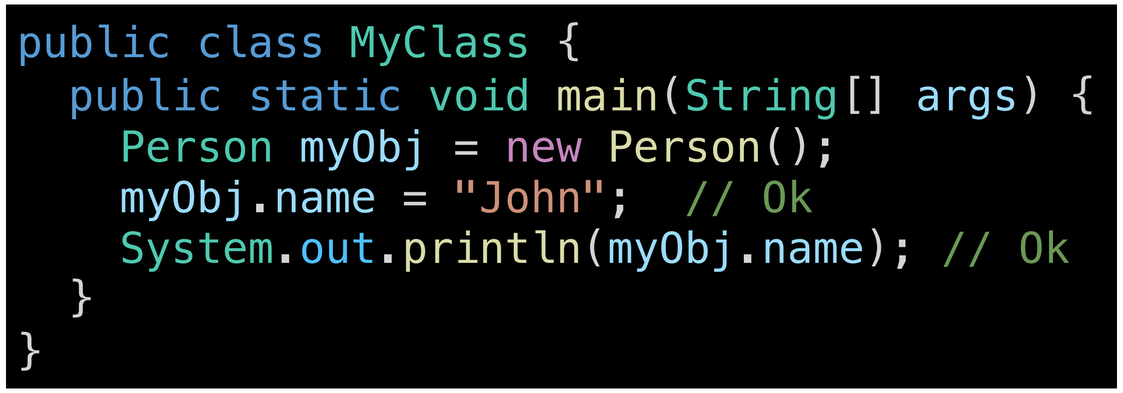
- If the variable was declared as public, we would expect the following output:



```
John
```



```
public class Person {
  public String name;

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```



```
public class MyClass {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.name = "John";  // Ok
    System.out.println(myObj.name); // Ok
  }
}
```

Get and Set

- Instead, we use the getName() and setName() methods to acccess and update the variable:

```
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

```
public class MyClass {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.setName("John"); // Set the value of the name variable to "John"
    System.out.println(myObj.getName());
  }
}

// Outputs "John"
```

Example of Encapsulation

```java
class EncapsulationDemo {
  private int empSSN;
  private String empName;
  private int empAge;
  // Getter and Setter methods
  public int getEmpSSN() {
    return empSsn;
  }
  public void setEmpSSN(int newValue) {
    empSsn = newValue;
  }
  public String getEmpName() {
    return empName;
  }
  public int getEmpAge() {
    return empAge;
  }
  public void setEmpAge(int newValue) {
    empAge = newValue;
  }
  public void setEmpName(String newValue) {
    empName = newValue;
  }
}
```

Example of Encapsulation

```java
public class EncapsTest {
  public static void main(String args[]) {
    EncapsulationDemo obj = new EncapsulationDemo();
    obj.setEmpName("Mario");
    obj.setEmpAge(32);
    obj.setEmpSSN(112233);
    System.out.println("Employee Name: " + obj.getEmpName());
    System.out.println("Employee SSN: " + obj.getEmpSSN());
    System.out.println("Employee Age: " + obj.getEmpAge());
  }
}
```

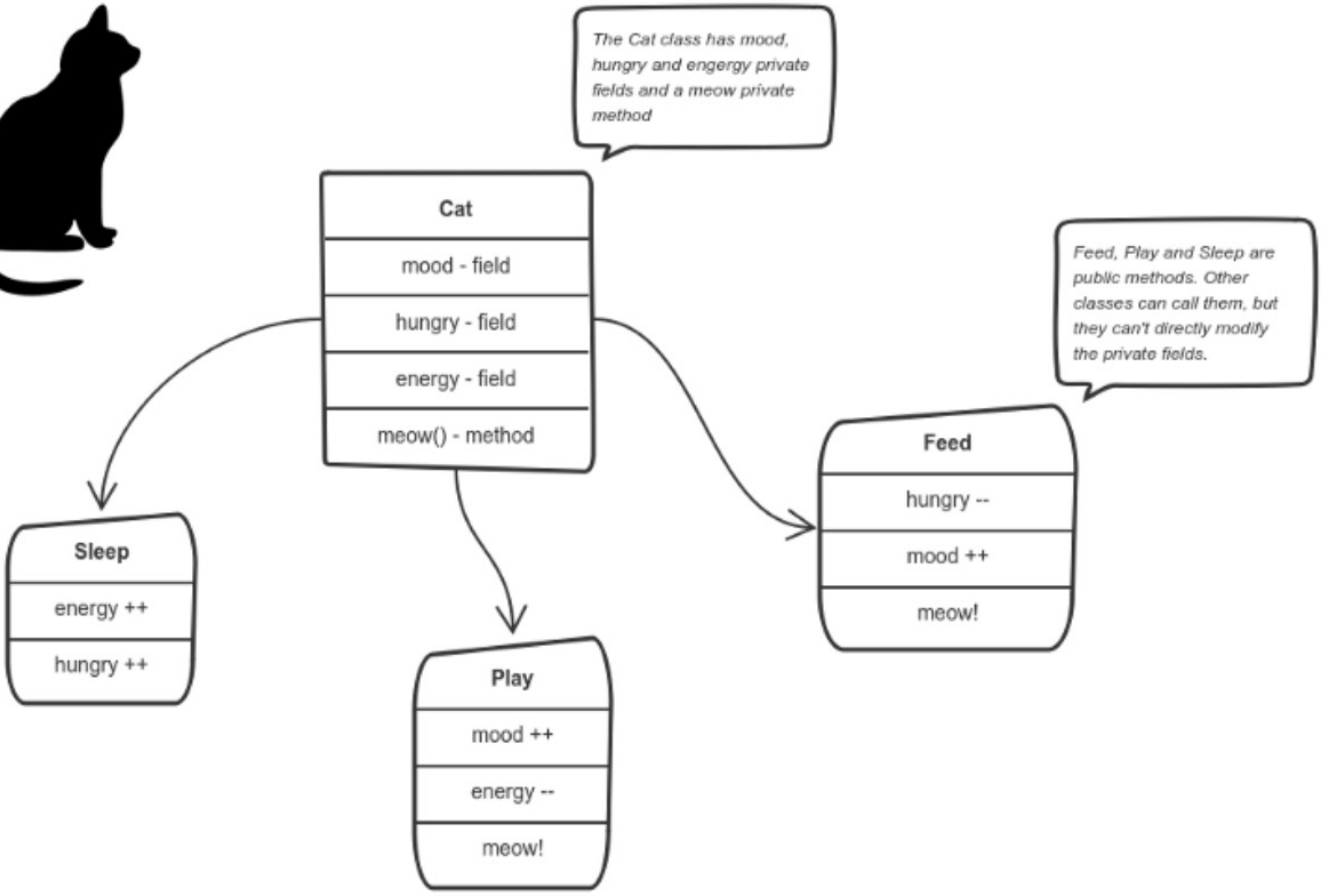
```
Output:

Employee Name: Mario
Employee SSN: 112233
Employee Age: 32
```

Why Encapsulation?

- Better control of class attributes and methods
- Class attributes can be made **read-only** (if you only use the get method), or **write-only** (if you only use the set method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

Lab04

- Design and create the Cat class according to the diagram as shown in the next slide. The cat has a private attribute mood, hungry, energy, and the private method meow() to display text "meow!!". The methods sleep(), play(), and feed() are public method. Create CatRun class to create at least 3 instance of Cat, and each instance are random call sleep(), play(), and feed() differently for 10 steps. After that, the program displays all status of mood, hungry, energy for every cat instance.



The Cat class has mood, hungry and engergy private fields and a meow private method

Feed, Play and Sleep are public methods. Other classes can call them, but they can't directly modify the private fields.

Reference

- 6 OOP Concepts in Java with examples [2020] · Raygun Blog https://raygun.com/blog/oop-concepts-java/ Accessed: 2020-07-04
- Encapsulation in Java – GeeksforGeeks https://www.geeksforgeeks.org/encapsulation-in-java/ Accessed: 2020-07-04
- ava Encapsulation and Getters and Settershttps://www.w3schools.com/java/java_encapsulation.asp Accessed: 2020-07-05
- How to explain object-oriented programming concepts to a 6-year-oldhttps://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/ Accessed: 2020-07-05
- Constructors in Java - A complete study!!https://beginnersbook.com/2013/03/constructors-in-java/ Accessed: 2020-07-05
- Visibility of Variables and Methods - Learning Java, 4th Edition [Book]https://www.oreilly.com/library/view/learning-java-4th/9781449372477/ch06s04.html Accessed: 2020-07-10
- new operator in Java - GeeksforGeekshttps://www.geeksforgeeks.org/new-operator-java/?ref=lbp Accessed: 2020-07-11