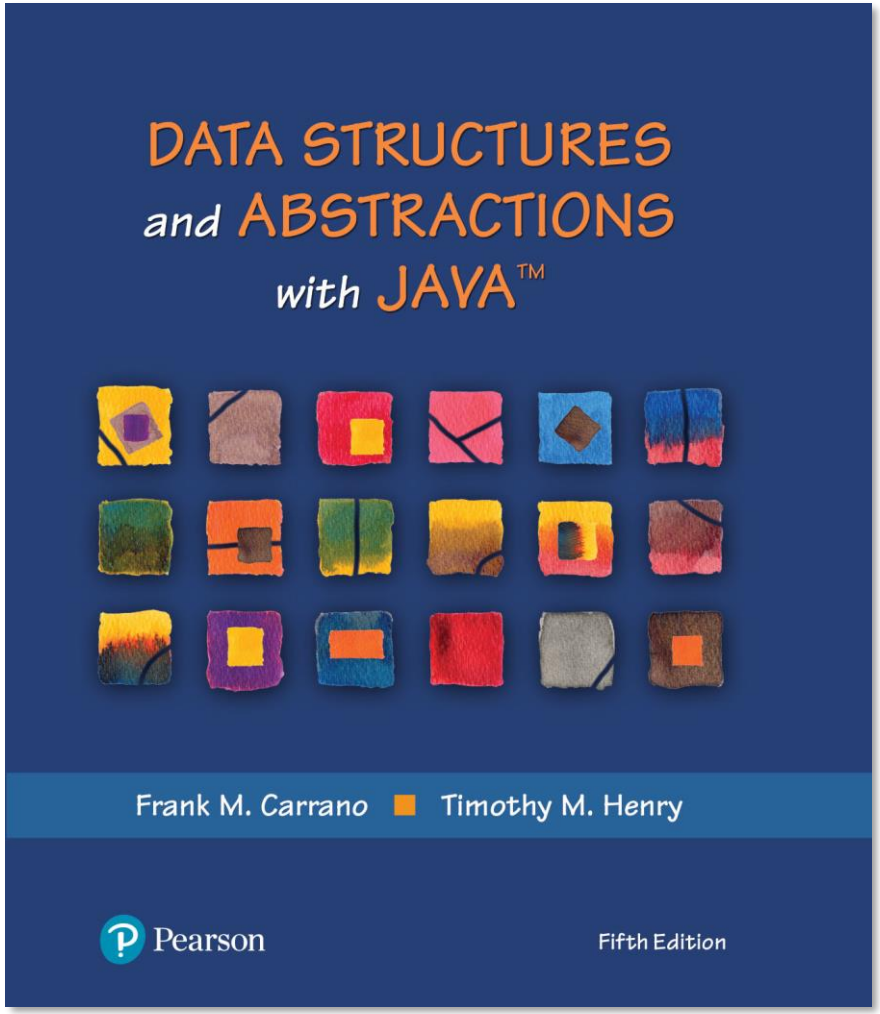
# Data Structures and Abstractions with Java™

5th Edition




## Java Interlude 7

# Inheritance and Polymorphism

# When to Use Inheritance

- Given the class **`VectorStack`**

```
public final class VectorStack<T> implements StackInterface<T>
{
  private  Vector<T> stack;
 . . .
```

- Use inheritance to derive **`VectorStack`** from **`Vector`**

```
public final class VectorStack<T> extends Vector<T>
                                  implements StackInterface<T>
{ . . .
```

# When to Use Inheritance

- Resulting class has all methods of Vector in addition to those in StackInterface
- However, these Vector methods enable a client to add or remove entries anywhere within the stack,
  - Thus violating the premise of the ADT stack.
  - Instead of a stack, we would have an enhanced vector
- Since no “is-a” relationship, should not use inheritance this way

# When to Use Inheritance

- Security issues
- Limit your use of inheritance
- Realize that a superclass can affect the behavior of a subclass

# Protected Access

- You can use the access modifier protected for methods and data fields.
- Method or data field modified by protected can be accessed by name only within:
  - Its own class definition C
  - Any class derived from C
  - Any class within the same package as C

# Access Modifiers

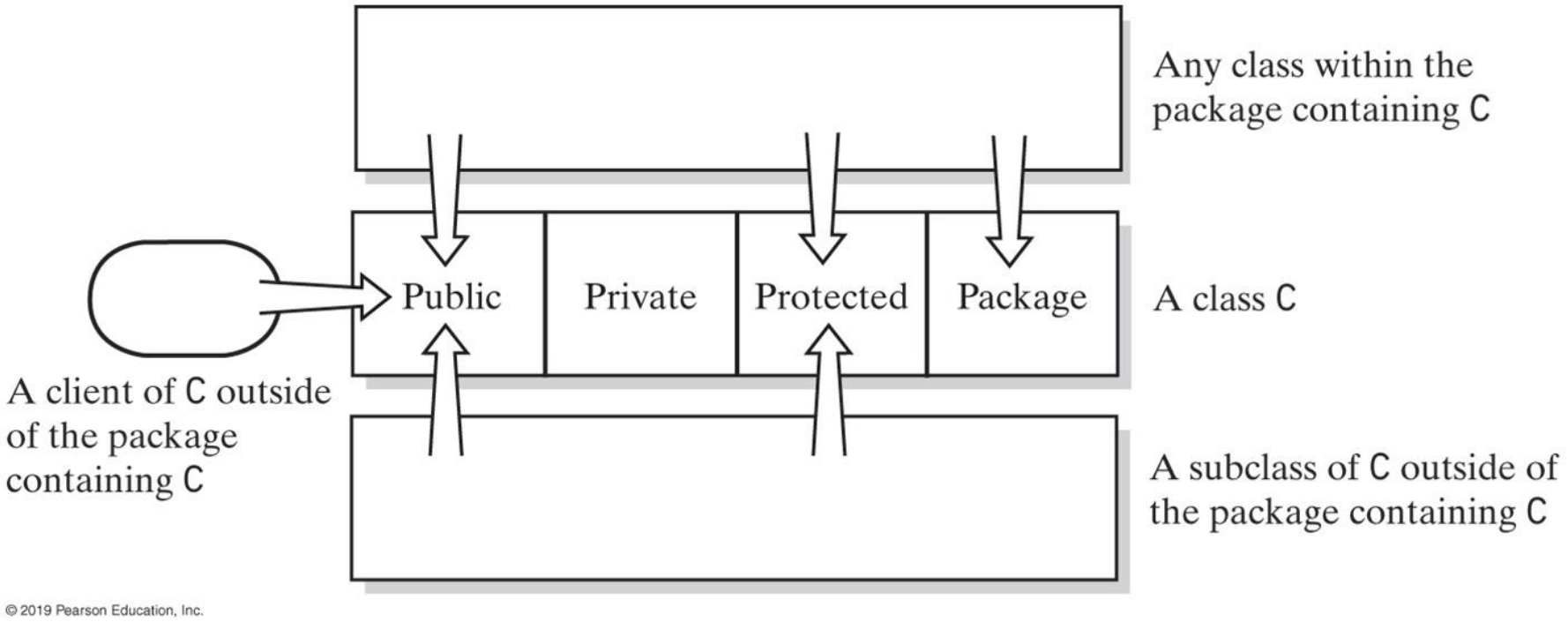



**FIGURE J7-1 Public, private, protected, and package access of the data fields and methods of class C**

# Abstract Classes and Methods

- An abstract class will be the superclass of another class
- Thus, an abstract class is sometimes called an abstract superclass.
- Declare abstract method by including reserved word abstract in header

```
public abstract void display();
```

# Abstract Classes and Methods

- Abstract method cannot be **`private`**, **`static`**, or **`final`**.
- Class with at least one abstract method must be declared as an abstract class
  - Abstract methods can appear only within an abstract class.
- Constructors cannot be abstract

# Polymorphism

- Polymorphism allows same program instruction to mean different things in different contexts
  - Context (which version of a method) determined at run time
- Java uses an object's dynamic type, not its name, to see which method to invoke

# Polymorphism

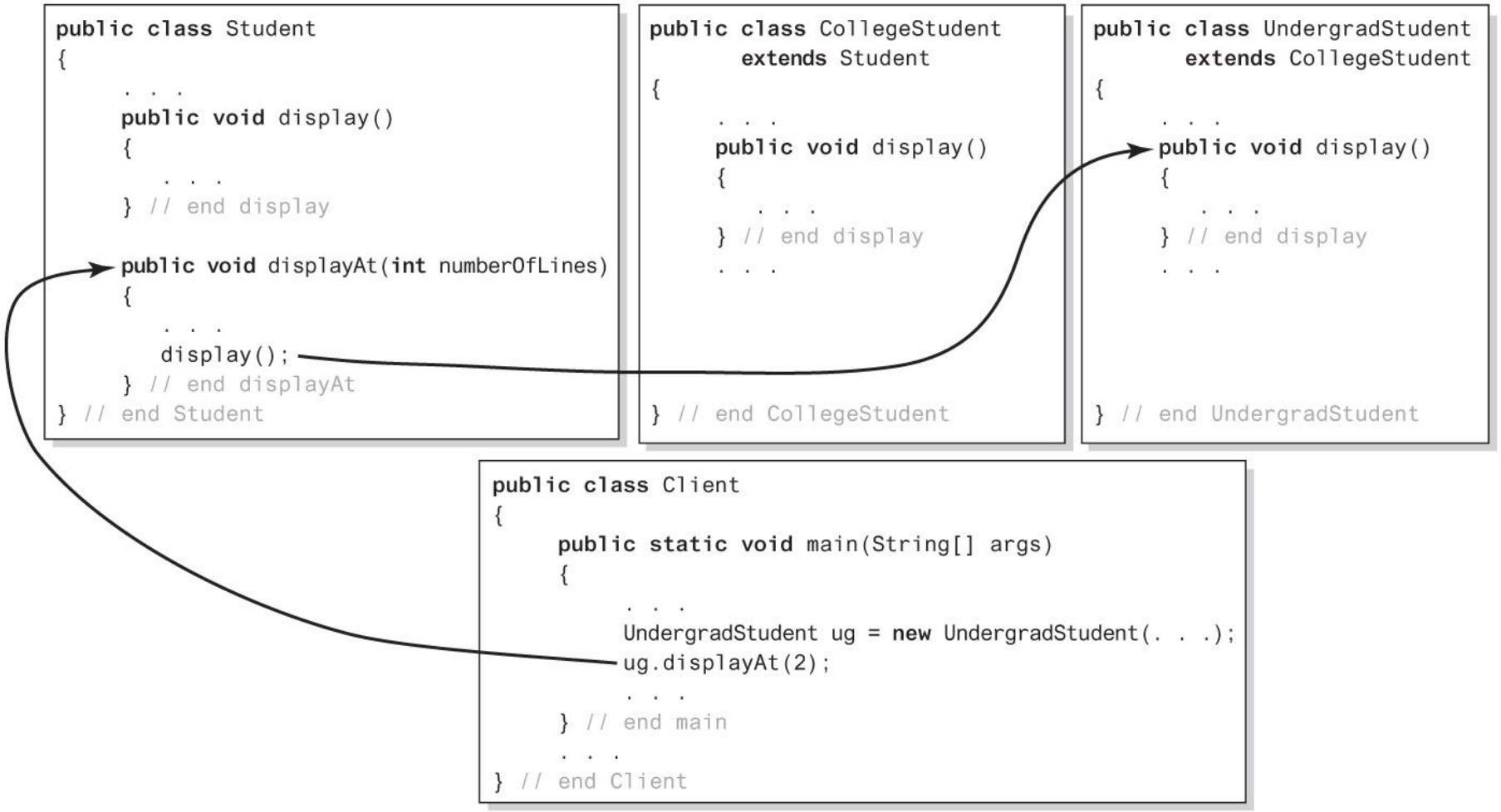





**FIGURE J7-2 The method displayAt calls the correct version of display**

# Polymorphism

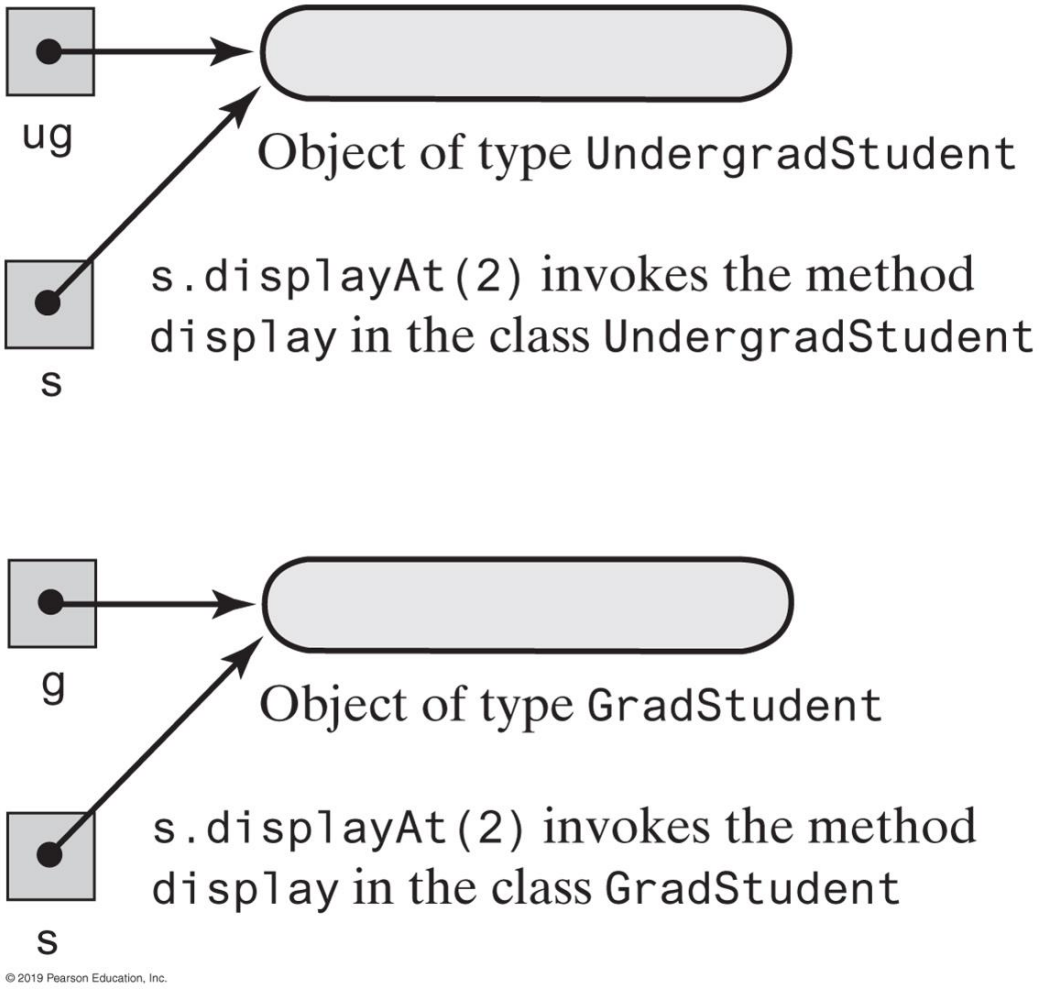



**FIGURE J7-3 An object, not its name, determines its behavior**

# End

# Java Interlude 7