PROGRAMMING 2

Gaza University

Musbah j. Mousa

ABSTRACT CLASS

- *An abstract class cannot be used to create objects. An abstract class can contain abstract methods that are implemented in concrete subclasses.*

In the inheritance hierarchy, classes become more specific and concrete *with each new subclass.*

If you move from a subclass back up to a superclass, the classes become more general and less specific. Class design should ensure a superclass contains common features of its subclasses. Sometimes, a superclass is so abstract it cannot be used to create any specific instances. Such a class is referred to as an *abstract class*.

POINTS ABOUT ABSTRACT CLASSES

- An abstract method cannot be contained in a nonabstract class. If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be defined as abstract. In other words, in a nonabstract subclass extended from an abstract class, all the abstract methods must be implemented. Also note abstract methods are nonstatic.
- An abstract class cannot be instantiated using the **new** operator, but you can still define its constructors, which are invoked in the constructors of its subclasses. For instance, the constructors of **GeometricObject** are invoked in the **Circle** class and the **Rectangle** class.

POINTS ABOUT ABSTRACT CLASSES

- A class that contains abstract methods must be abstract. However, it is possible to define an abstract class that doesn’t contain any abstract methods. This abstract class is used as a base class for defining subclasses.
- A subclass can override a method from its superclass to define it as abstract. This is *very unusual*, but it is useful when the implementation of the method in the superclass becomes invalid in the subclass. In this case, the subclass must be defined as abstract.

POINTS ABOUT ABSTRACT CLASSES

- A subclass can be abstract even if its superclass is concrete. For example, the **Object** class is concrete, but its subclasses, such as **GeometricObject**, may be abstract.
- You cannot create an instance from an abstract class using the **new** operator, but an abstract class can be used as a data type. Therefore, the following statement, which creates an array whose elements are of the **GeometricObject** type, is correct:

GeometricObject[] objects = **new** GeometricObject[**10**];

ABSTRACT CLASSES

- 1) Write an abstract class Shape
– Data members: numSides
– Constructor: initialize numSides
– Concrete method: get method for numSides
– Abstract methods: getArea(), getPerimeter()
2) Write a concrete subclass Rectangle
– Data members: width, height
3) Write a concrete subclass RtTriangle
– Data members: width, height
4) In another class, write a main method to define a Rectangle and a Triangle.

INTERFACES

- *An interface is a class-like construct for defining common operations for objects.*
- Interface is similar to an abstract class, but its intent is to specify common behavior for objects of related classes or unrelated classes.

INTERFACES

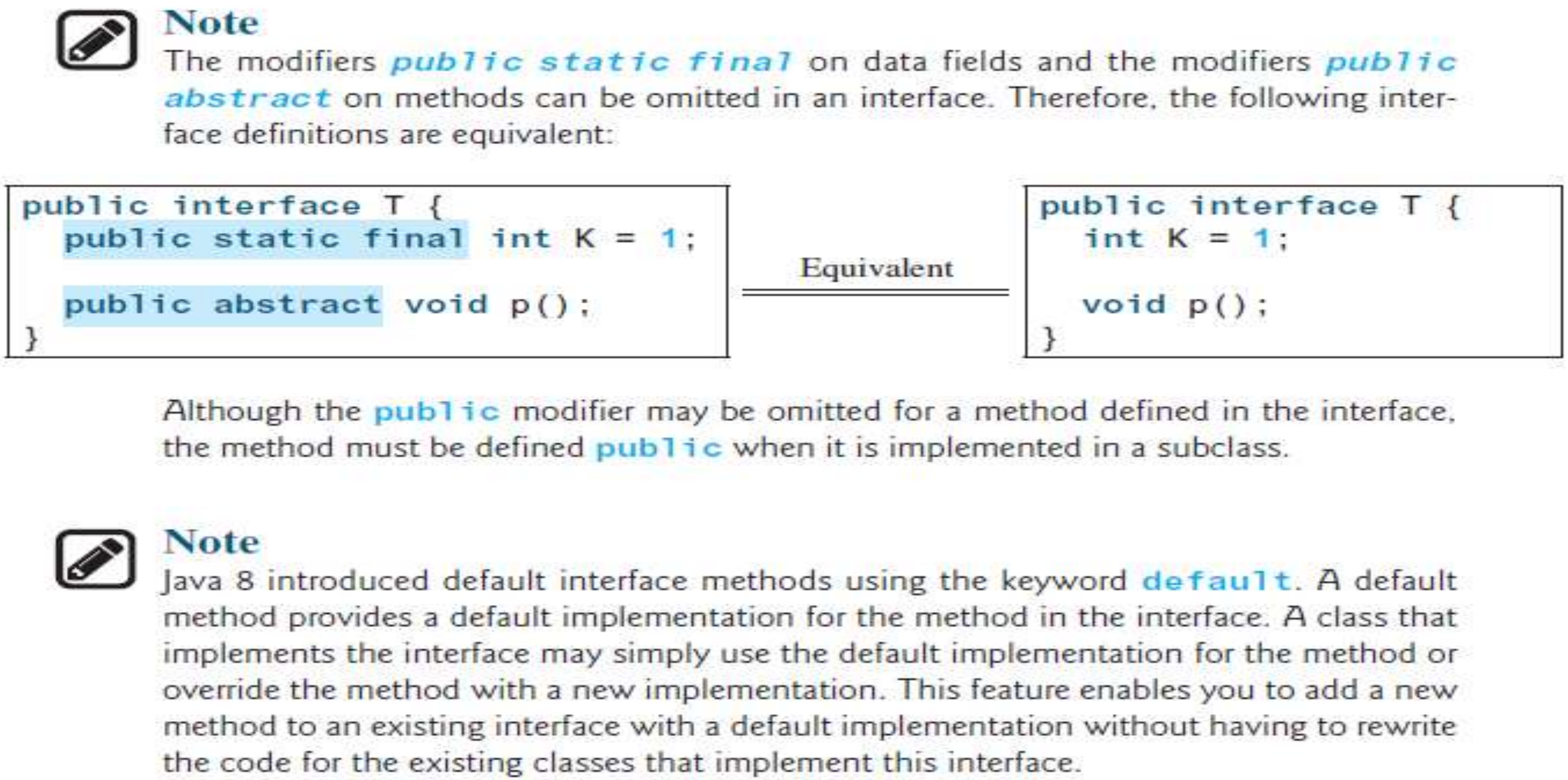
- An interface declares (describes) methods but does not supply bodies for them .
- All the methods are implicitly public and abstract .
- You cannot instantiate an interface .
- An interface may also contain constants (final variables) .

INTERFACES

- An interface is created with the following syntax

```
 modifier interface interfaceName
{
//constants
//method signatures
 }
```

INTERFACES



Note

The modifiers *public static final* on data fields and the modifiers *public abstract* on methods can be omitted in an interface. Therefore, the following interface definitions are equivalent:

```
public interface T {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

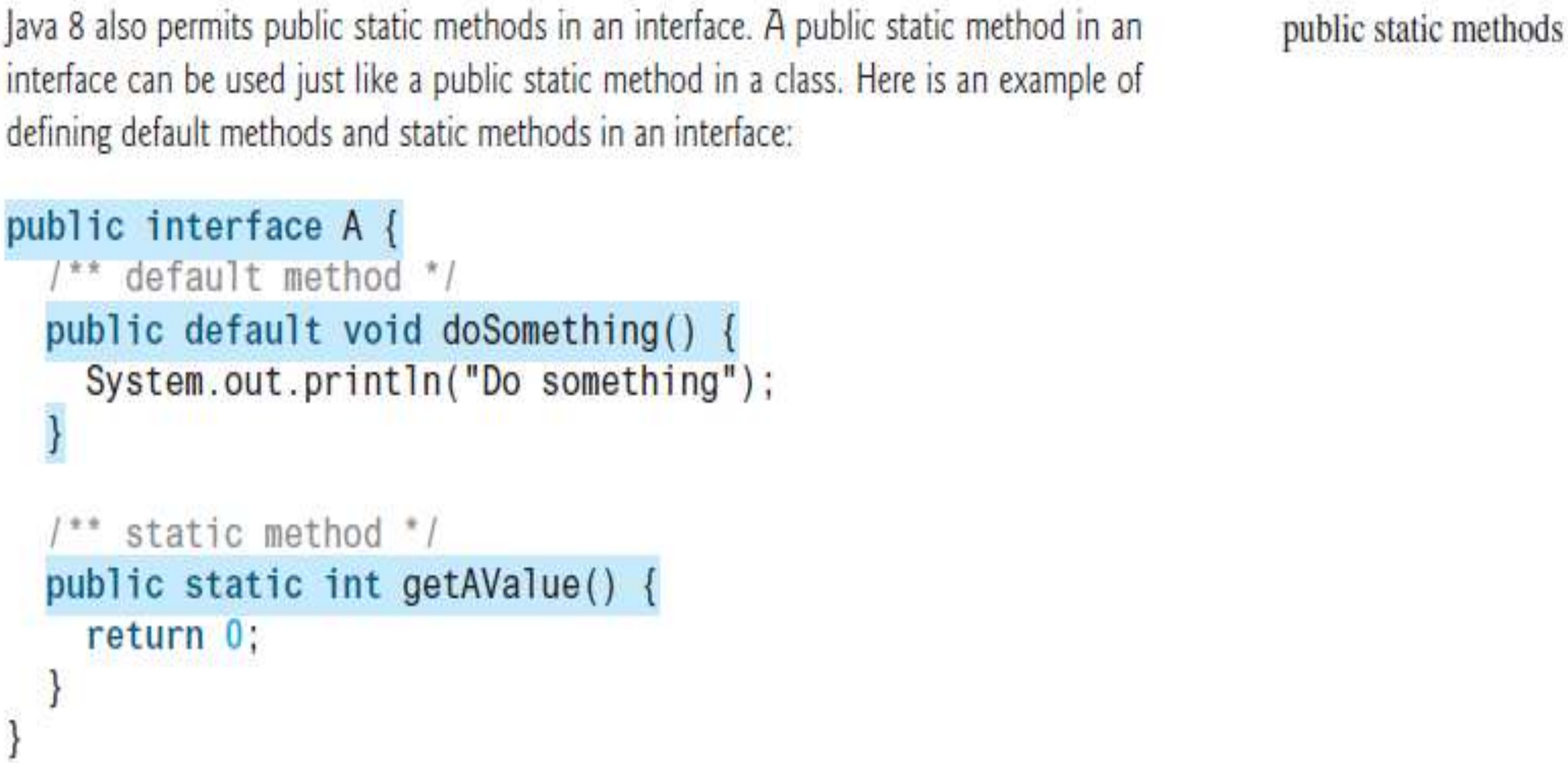
```
public interface T {
  int K = 1;

  void p();
}
```

Although the `public` modifier may be omitted for a method defined in the interface, the method must be defined `public` when it is implemented in a subclass.

Note

Java 8 introduced default interface methods using the keyword `default`. A default method provides a default implementation for the method in the interface. A class that implements the interface may simply use the default implementation for the method or override the method with a new implementation. This feature enables you to add a new method to an existing interface with a default implementation without having to rewrite the code for the existing classes that implement this interface.

INTERFACES



public static methods

Java 8 also permits public static methods in an interface. A public static method in an interface can be used just like a public static method in a class. Here is an example of defining default methods and static methods in an interface:

```
public interface A {
  /** default method */
  public default void doSomething() {
    System.out.println("Do something");
  }

  /** static method */
  public static int getAValue() {
    return 0;
  }
}
```

INTERFACES

- An interface can extend other interfaces with the following syntax:

modifier interface interfaceName extends comma-delimited-list-of- interfaces

{

//constants

/method signatures

}

- Obviously, any class which implements a “sub-interface” will have to implement each of the methods contained in it’s “super-interfaces”

INTERFACES

- You extend a class, but you implement an interface .
- A class can only extend (subclass) one other class, but it can implement as many interfaces as you like .
- Example:

```
class MyListener implements KeyListener, ActionListener
 {
…
 }
```

- When you say a class implements an interface, you are promising to define all the methods that were declared in the interface.

WHAT ARE INTERFACES FOR?

- Reason 1: A class can only extend one other class, but it can implement multiple interfaces .
- This lets the class fill multiple “roles” .
- Example:

```
class MyApplet extends Applet implements ActionListener, KeyListener
{
…
 }
```

- Reason 2: You can write methods that work for more than one kind of class .

QUESTIONS?