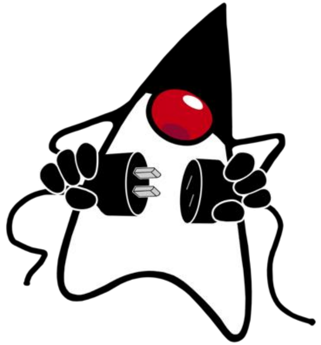
Packages and Interfaces

CO7005 Software Development Techniques



Dr Stuart Cunningham
s.cunningham@chester.ac.uk

*"Java provides **packages** , a layer of structure that groups classes into functional units. Packages provide a naming convention for organizing classes and a second tier of organizational control over the visibility of variables and methods in Java applications…. This lends itself to building reusable components that work together in a system."*

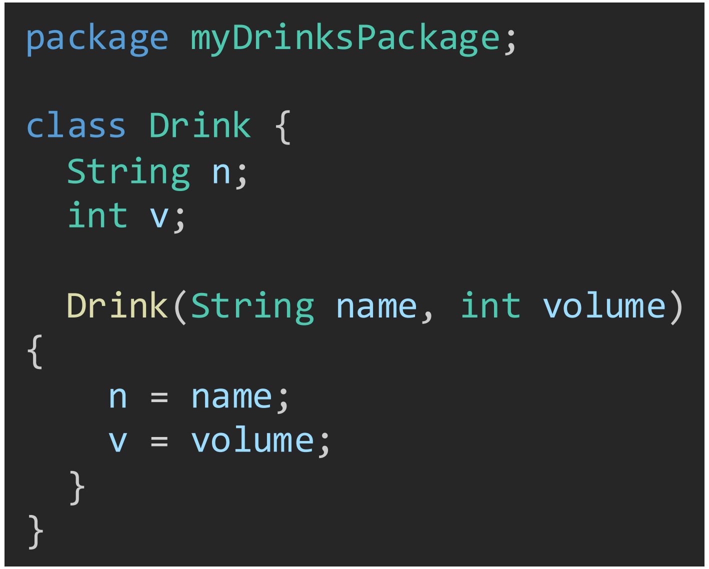
(Loy 2020)

Packages

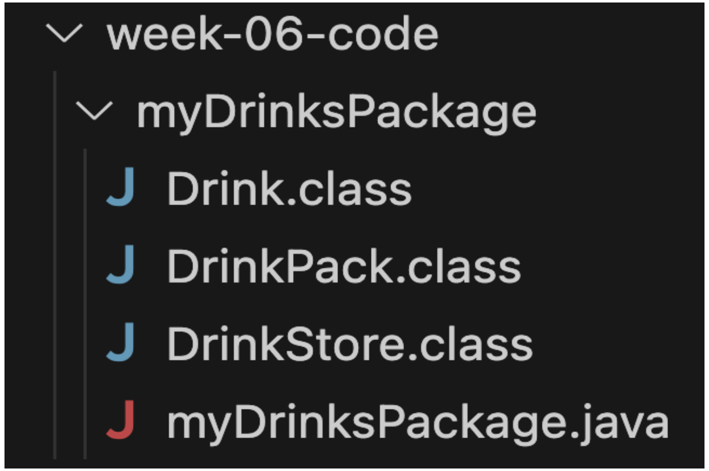
- Organise a *collection* of related classes
- Further supporting class *encapsulation*
    - Access can be limited to *package scope*
- Manages naming of related classes
    - E.g., Drink, DrinkStore, DrinkPack, etc.
- Classes in a package have the package name attached, avoiding conflicts
- Declare with `package myDrinksPackage;`

Packages

- Package classes are typically stored in a named folder
- Supports hierarchical organisation (e.g. sub-folders)
- Compiling the package results in its classes being created
- Accessed by other programs using `import myDrinksPackage.*;`
- Public classes must be in separate files and appropriate methods (constructor, get, set, etc.) made public



```
package myDrinksPackage;

class Drink {
  String n;
  int v;

  Drink(String name, int volume)
{
    n = name;
    v = volume;
  }
}
```



```
∨ week-06-code
  ∨ myDrinksPackage
    J Drink.class
    J DrinkPack.class
    J DrinkStore.class
    J myDrinksPackage.java
```

```java
package myDrinksPackage;

public class DrinkManufacturer {
  private String name;
  private String address;
  private String city;
  private int manuID;

  public DrinkManufacturer (String name,  String address, String city, int manuID) {
    this.name = name;
    this.address = address;
    this.city = city;
    this.manuID = manuID;
  }

  public String getDetails() {
    return name+"\n"+address+"\n"+city+"\n"+manuID;
  }
}
```

```java
import myDrinksPackage.DrinkManufacturer;

public class DrinkTest {
  public static void main(String[] args) {
    System.out.println("Drinks Test");
    System.out.println("-----------");
    DrinkManufacturer nestle = new DrinkManufacturer("Nestle", "Marston Lane", "Burton-on-Trent", 427);
    System.out.println(nestle.getDetails());
  }
}
```

```
>> java DrinkTest
Drinks Test
-----------
Nestle
Marston Lane
Burton-on-Trent
427
```

Packages and Access (Schildt 2022)

	`private`	`default`	`protected`	`public`
Visible within same class	Yes	Yes	Yes	Yes
Visible within same package by subclass	No	Yes	Yes	Yes
Visible within same package by non-subclass	No	Yes	Yes	Yes
Visible within different package by subclass	No	No	Yes	Yes
Visible within different package by non-subclass	No	No	No	Yes

Default Packages

- *Packages* provide a way to bundle related, useful classes
- A variety of packages are part of Java API
  - For example, java.util.Scanner

Package	Description
java.io	Provides for system input and output through data streams, serialization and the file system.
java.math	Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal).
java.time	The main API for dates, times, instants, and durations.
java.util	Contains the collections framework, some internationalization support classes, a service loader, properties, random number generation, string parsing and scanning classes, base64 encoding and decoding, a bit array, and several miscellaneous utility classes.

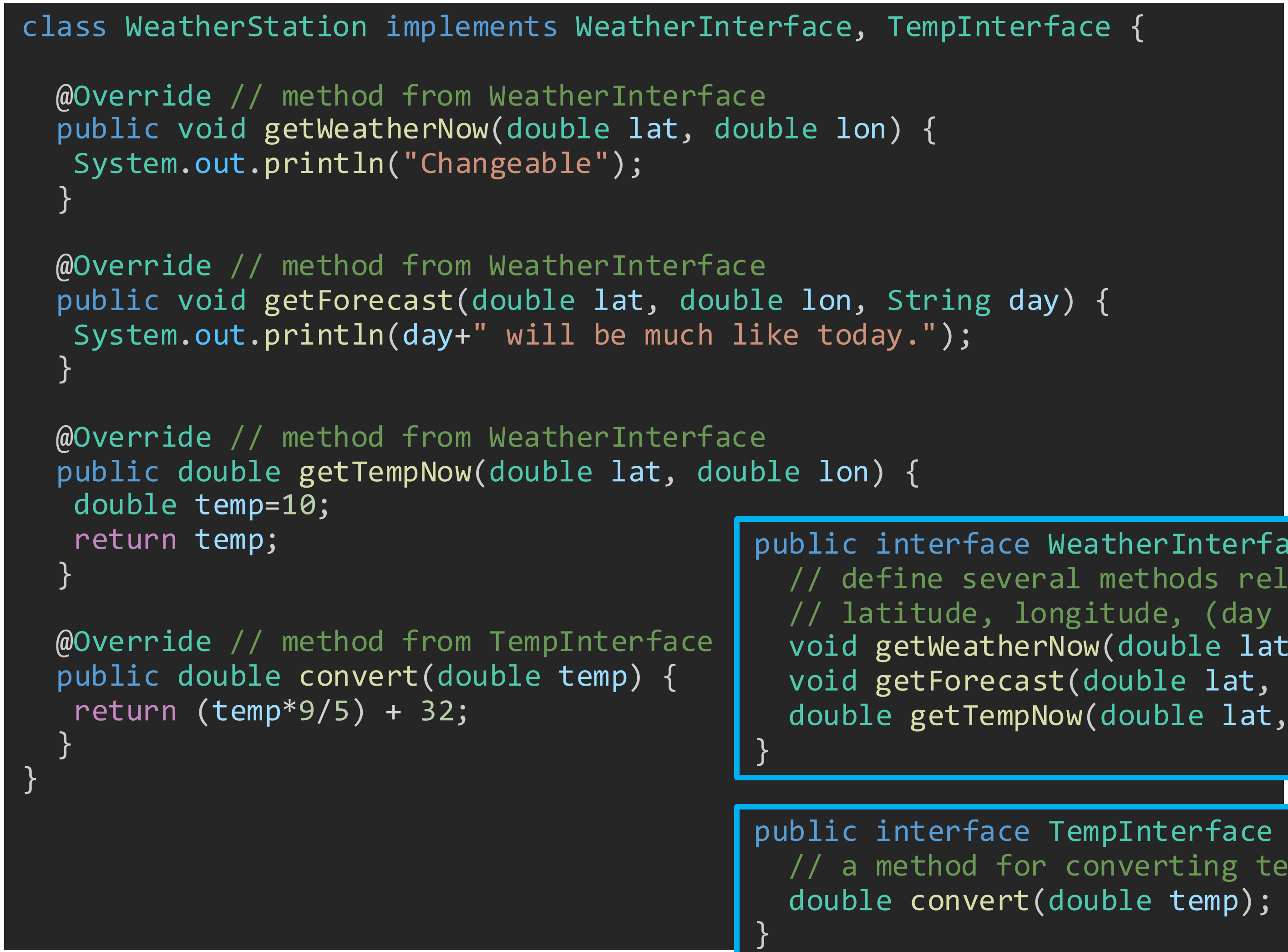
Source: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/module-summary.html

*“**Interfaces** define and standardize the ways in which things such as people and systems can interact with one another…The interface specifies what operations [a radio] must permit users to perform but does not specify how the operations are performed.”*

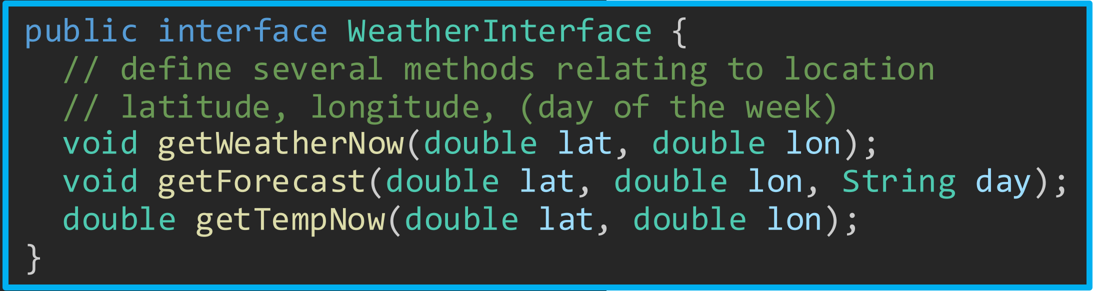
(Deitel & Deitel 2018)

Interfaces

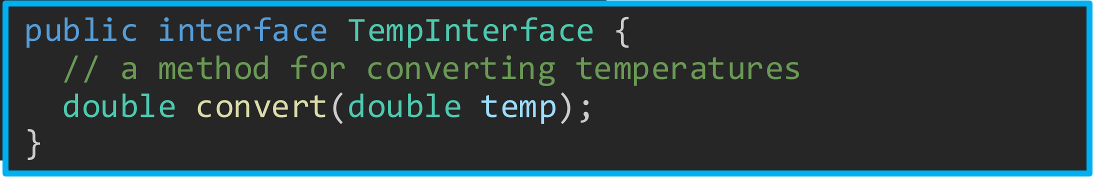
- Like *abstract* classes and methods with *overriding*
- What can be done with classes (without knowing *how*)
- *Interfaces* operate similarly but without inheritance
- Overridden methods must be included and public
- Classes can *implement* many interfaces
  - Unlike inheritance, where a sub-class has only one super-class
- Interfaces may be *implemented* by multiple classes
- Implementing classes can define their own methods



```java
class WeatherStation implements WeatherInterface, TempInterface {

   @Override // method from WeatherInterface
   public void getWeatherNow(double lat, double lon) {
    System.out.println("Changeable");
   }

   @Override // method from WeatherInterface
   public void getForecast(double lat, double lon, String day) {
    System.out.println(day+" will be much like today.");
   }

   @Override // method from WeatherInterface
   public double getTempNow(double lat, double lon) {
    double temp=10;
    return temp;
   }

   @Override // method from TempInterface
   public double convert(double temp) {
    return (temp*9/5) + 32;
   }
}
```



```java
public interface WeatherInterface {
  // define several methods relating to location
  // latitude, longitude, (day of the week)
  void getWeatherNow(double lat, double lon);
  void getForecast(double lat, double lon, String day);
  double getTempNow(double lat, double lon);
}
```



```java
public interface TempInterface {
  // a method for converting temperatures
  double convert(double temp);
}
```

Interfaces

- Further support *encapsulation*, hiding details of attributes
- Permit a *variety* of methods, with multiple interfaces being re-used by any program
- Default behaviour *can* be defined (≥JDK 8) – but often not in practice
  - Sometimes referred to as an *extension* method
  - Useful when *updated* interfaces are rolled out to prevent existing implementations from breaking
  - Methods that aren’t needed can also be ignored in implementation
- `default float getSomeNumber() { }`

Interfaces

- *Interface variables* (`public static final`) can be declared and are useful *constants* in large programs / multiple classes
- But interfaces do not have *instance variables*
- Interfaces can *inherit* from other interfaces (like classes) by *extending* them

```java
public interface WeatherConst {
   static final int MINTEMP = -35;
   static final int MAXTEMP = 65;
   static final float VER = 2.6F;
}
```

```java
class WeatherStation implements
WeatherInterface, TempInterface,
WeatherConst {
// code here as before
}
```

```java
public class WeatherProgram {
  public static void main(String[] args) {
    WeatherStation binks = new WeatherStation("Thursday");
    //get forecast and info from the WeatherStation
    System.out.print("The weather today is ");
    binks.getWeatherNow(53.199954,-2.898662);
    double tempNowC = binks.getTempNow(53.199954,-2.898662);
    double tempNowF = binks.convert(tempNowC);
    System.out.print("Temperature is "+tempNowC+" \u00B0C");
    System.out.println(" or "+tempNowF+" \u00B0F");
    binks.getForecast(53.199954,-2.898662,"Saturday");
    System.out.println("---------------");
    // get constant variables from WeatherConst interface
    // notice use of the class name (not instance) when accessing
    System.out.print("Limit: "+WeatherStation.MINTEMP+"\u00B0C to ");
    System.out.println(WeatherStation.MAXTEMP+"\u00B0C");
  }
}
```

Interfaces – Static Methods

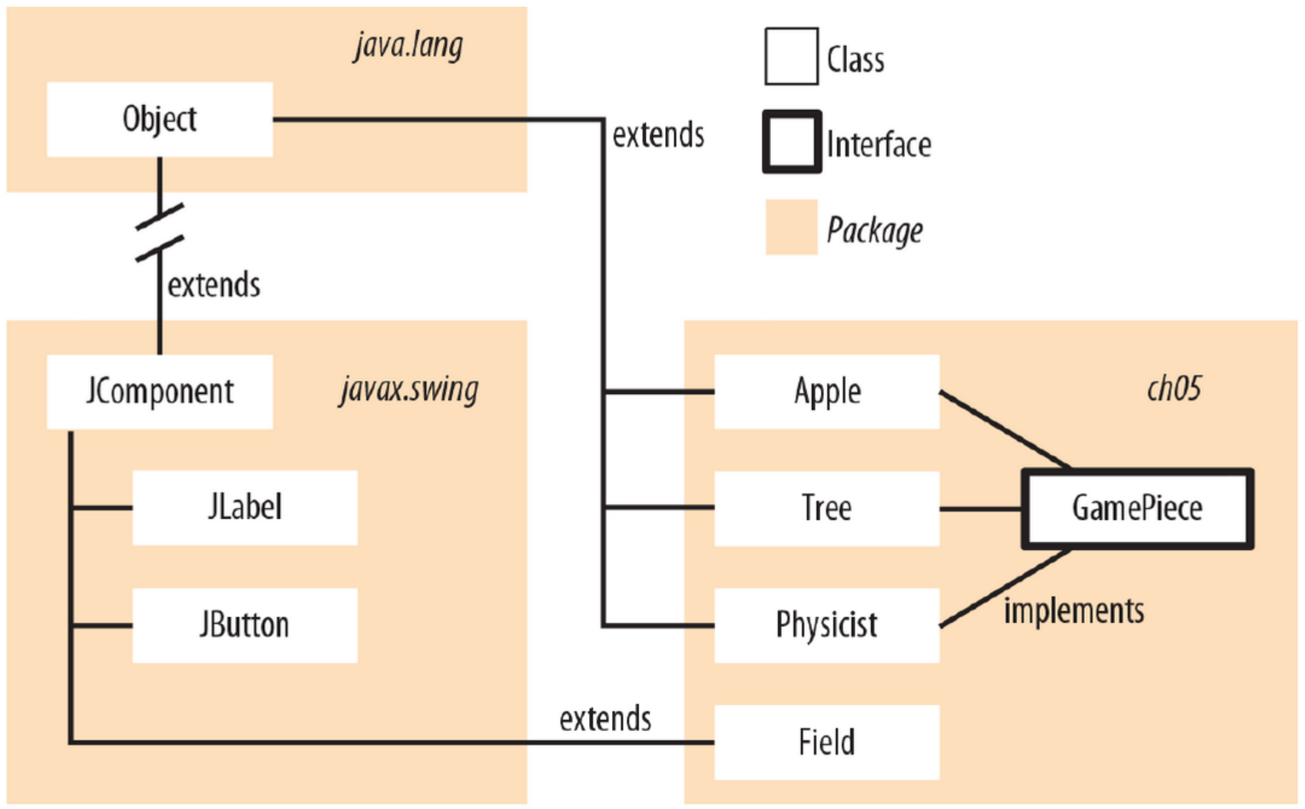
- *Static* methods can be declared in interfaces
- Directly referenced - class doesn't *implement* the interface

```
public interface StaticMethodInterface {
  static String HelloWorld() {
    return "Hello, World!";
  }
}
```

```
public class HelloStatic {
  public static void main(String[] args) {
    String message = StaticMethodInterface.HelloWorld();
    System.out.println(message);
  }
}
```

```
java HelloStatic
Hello, World!
```

Class, interface and package overview (Loy 2020)



References

Deitel, P. J., & Deitel, H. M. (2018). *Java: how to program : early objects* (Global). Pearson.

Loy, M. (2020). *Learning Java: an introduction to real-world programming with Java* (Fifth). O'Reilly.

Schildt, H. (2022). *Java: a beginner's guide* (9th ed.). McGraw Hill.