



Creating and Using Packages

- Package
 - A named collection of classes
 - Easily imports related classes into new programs
 - Encourages other programmers to reuse software
 - Helps avoid naming conflicts or collisions
 - Gives every package a unique name



Creating and Using Packages (cont)

- Compile the file to place in a package
 - We can use the export option, however, the parameters need to be changed
- We are going to demonstrate them in the demo
- Package-naming convention
 - Use your Internet domain name in reverse order



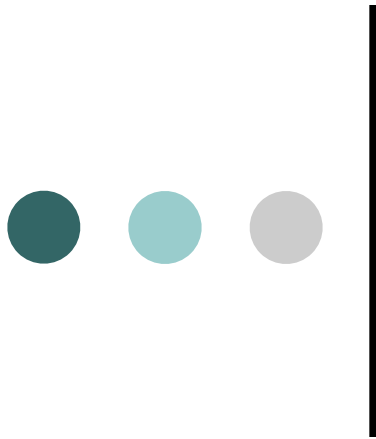
Creating and Using Packages (cont)

- Java ARchive (JAR) file
 - A package or class library is delivered to users as a JAR file
 - Compresses and stores data
- Reduces the size of archived class files
 - Based on the Zip file format
- Demo



Creating and Using Packages (cont)

- After the compiled jar file has been created, we can use it in a separate project
- To do so, we have to include the jar file in the classpath of the project
 - All the public classes will be available in the current project
 - Make sure not to delete the jar file, otherwise, the project won't be able to use it
- What happens to the protected instance variables?
- Demo



Chapter 10 – Interfaces

Dr Kafi Rahman

Assistant Professor @CS

Truman State University



The Comparable Interface

- Because the `BankAccount` class implements the `Comparable` interface, you can sort an array of bank accounts with the `Arrays.sort` method:
 - `BankAccount[] accounts = new BankAccount[3];`
 - `accounts[0] = new BankAccount(10000);`
 - `accounts[1] = new BankAccount(0);`
 - `accounts[2] = new BankAccount(2000);`
 - `Arrays.sort(accounts);`
- Now the `accounts` array is sorted by increasing balance.
- The `compareTo` method checks whether another object is larger or smaller.



Self Check 10.11

- How can you sort an array of Country objects by increasing area?
- Answer: Have the Country class implement the Comparable interface, as shown below, and call Arrays.sort.

```
public class Country implements Comparable
{
    . . .
    public int compareTo(Object otherObject)
    {
        Country other = (Country) otherObject;
        if (area < other.area) { return -1; }
        if (area > other.area) { return 1; }
        return 0;
    }
}
```



Self Check 10.13

- The Rectangle class does not implement any interface
- Can we use the Arrays.sort method to sort an array of Rectangle objects?
 - Answer: No. The Rectangle class does not implement the Comparable interface.



Self Check 10.14

- Write a method `max` that finds the larger of any two `Comparable` objects.
- Answer:

```
public static Comparable max(Comparable a, Comparable b)
{   if (a.compareTo(b) > 0) { return a; }
    else { return b; }
}
```



Self Check 10.15

- Write a call to the method of Self Check 14 that computes the larger of two bank accounts, then prints its balance.
- Answer:

```
BankAccount larger = (BankAccount) max(first, second);  
System.out.println(larger.getBalance());
```

- Note that the result must be cast from Comparable to BankAccount so that we can invoke the getBalance method.



Using Interfaces

- We want to determine average of the area of a number of classes
 - However, each class has their own way of calculating the area
- We can create an interface and implement the interface for each class:

```
public interface Measurable {  
    public double calcArea();  
}
```



Using Interfaces (cont)

- For example, the Circle class can implement the Measurable interface in its definition

```
public class Circle implements Measurable {  
  
    private double radius;  
  
    public Circle(double r) {  
        this.radius = r;  
    }  
  
    public double calcArea() {  
        return Math.PI * Math.pow(this.radius, 2.0);  
    }  
}
```



Using Interfaces (cont)

- For example, the Rectangle class can implement the Measurable interface in its definition

```
public class Rectangle implements Measurable {  
  
    private double width, length;  
  
    public Rectangle(double w, double l) {  
        this.width = w;  
        this.length = l;  
    }  
  
    public double calcArea() {  
        return this.width * this.length;  
    }  
}
```



Using Interfaces (cont)

- Therefore, determining the average of the area of the objects of these classes would be straightforward

```
public static double calcAverage(Measurable[] measArray) {  
    double average = 0;  
    if (measArray != null && measArray.length == 0)  
        return average;  
  
    for (Measurable myObj : measArray) {  
        average += myObj.calcArea();  
    }  
  
    return average / measArray.length;  
}
```



Please let me know if you have any questions.



Questions?