Contents

1. Classes and Objects
2. Inheritance
3. Interfaces
4. Exceptions and Error Handling
5. Intro to Concurrency
6. Concurrency in Java
7. Graphics and Animation
8. Applets
Chapter 3: Interfaces

After this chapter you will be able to:

- Understand the motivation for interfaces
- Write classes that implement interfaces
- Use interfaces to write generic code
Review Of Inheritance

• **Question:** What does inheritance really mean?

• We used inheritance to do three different things in the previous chapter.

• The three things are?
Answer

• Inheritance is used for:
  – **Subtyping:** If \( B \) is a subclass of \( A \) then a \( B \) object can be used in any context where an \( A \) is required
  – **Inheritance of implementation:** Subclasses inherit the instance variables and methods of superclasses
  – **Inheritance of interface:** Using abstract classes, inheritance was used to force subclasses to provide a certain interface
Conclusion

• Inheritance is a complex and heavyweight mechanism

• When viewing an inheritance hierarchy it may not be apparent why inheritance is being used

• The future trend in programming languages will be towards simpler (lightweight) mechanisms
Interfaces

• The Java interface is a simpler mechanism used only to require classes to support a certain interface

• The classes supporting an interface are not necessarily related by inheritance

**Benefit:** Can write generic code with respect to classes supporting the same interface
Java API

• The Java API consists of classes and interfaces

```java
public interface java.lang.Runnable {
    public abstract void run();
}
```

Interface Runnable is used for multithreading
Interface Definition

• Use keyword `interface`

```java
interface Printable {
    void print ();
}
```

• An `interface` may contain a sequence of method specifications (no body)
• The method specifications are implicitly public
Implementing An Interface

• A class conforms to an interface by using keyword implements and by defining the required methods

```java
interface Printable {
    void print ();
}

class Person implements Printable {
    private String name;

    public Person (String name) {
        this.name = name;
    }

    public void print () {
        System.out.println("My name is " + name);
    }
}
```
Compiler Error

- If a class implements an interface but does not define all the required methods, a compile-time error results.

```java
interface Printable {
    void print ();
}

class Person implements Printable {
    private String name;

    public Person (String name) {
        this.name = name;
    }

    /* no definition of print */
}
```
Interface Names

• Interface names may be used as data-types (just like class names)

```java
Printable p;
```

• Variable $p$ may be bound to any object of a class which implements interface `Printable`

```java
Printable p = new Person("ed");
```
Rolodex Example

- Can store different kinds of objects (People and Companies) in a rolodex if the objects support the interface `RolodexEntry`

```java
interface RolodexEntry {
    String getName();
    String getAddress();
    RolodexEntry getNext();
    void setNext(RolodexEntry next);
}
```
Class Rolodex

• Use interface RolodexEntry as the datatype of the items in the Rolodex

```java
public class Rolodex {
    private RolodexEntry list = null;

    public void add (RolodexEntry newItem) {
        if (list == null) { /* empty list */
            list = newItem;
            list.setNext(null);
        } else { /* new item becomes head */
            newItem.setNext(list);
            list = newItem;
        }
    }
}
...
Method print

- Method **print** traverses the linked list of Rolodex entries and calls **getName** and **getAddress**

```java
public class Rolodex {
    private RolodexEntry list = null;
    public void add (RolodexEntry newItem) {...}

    public void print () {
        for(RolodexEntry i = list; i != null; i = i.getNext())
            System.out.println(i.getName() + " " + i.getAddress());
    }
}
```

- This is generic code with respect to interface RolodexEntry
Class Person

• Class Person implements RolodexEntry

class Person implements RolodexEntry {
    private String firstName;
    private String lastName;
    private String address;
    private RolodexEntry next = null;

    public Person (String firstName, String lastName, String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
    }

    ...
}
class Person implements RolodexEntry {

    ... public String getName() {
        return firstName + " " + lastName;
    }

    public String getAddress() { return address; }

    public String getAge() { return address; }

    public RolodexEntry getNext() { return next; }

    public void setNext(RolodexEntry next) {
        this.next = next;
    }
}
Class Company

- Class Company implements RolodexEntry

```java
class Company implements RolodexEntry {
    private String name;
    private String address;
    private int numEmployees;

    private RolodexEntry next = null;

    public Company (String name, String address,
                    int numEmployees) {
        this.name = name;
        this.address = address;
        this.numEmployees = numEmployees;
    }

    ...
```
class Person implements RolodexEntry {

  ...

  public String getName() { return name; }
  public String getAddress() { return address; }
  public String getAge() { return address; }
  public RolodexEntry getNext() { return next; }
  public void setNext(RolodexEntry next) {
    this.next = next;
  }

}
Creating A Rolodex

• Can add **Person** and **Company** objects to the Rolodex

```java
public static void main (String [] args) {
    Rolodex r = new Rolodex();
    r.add(new Person("ed","smith","brooklyn");)
    r.add(new Company("Acme","NYC", 10000));
    r.print();
}
```

**Output**

```
Acme NYC ed smith brooklyn
```
Extending Interfaces

• An interfaces can extend another interface

```java
interface RolodexEntry {
    String getName();
    String getAddress();
    RolodexEntry getNext();
    void setNext(RolodexEntry next);
}

interface EntryWithDate extends RolodexEntry {
    String getDate();
}
```

• A class which implements EntryWithDate must define the methods in both interfaces
Interfaces And Inheritance

• If a class implements an interface then so do its subclasses

```java
class Employee extends Person {
    private String socSecurity;

    public Employee (String firstName, String lastName, 
                     String address, String socSecurity) {
        super(firstName, lastName, address);
        this.socSecurity = socSecurity;
    }
}
```

• Class Employee implicitly implements interface RolodexEntry
Adding Employees

• Can add Employees to the Rolodex

```java
public static void main(String[] args) {
    Rolodex r = new Rolodex();
    r.add(new Person("ed","smith","brooklyn");
    r.add(new Company("Acme","NYC", 10000));
    r.add(new Employee("fran","kennedy","queens", "031-54-4123");
    r.print();
}
```

Output

```
fran kenedy queens
Acme NYC
ed smith brooklyn
```
Interfaces and final variables

- Interfaces may also declare final variables
- final variables are constants and must be initialized

```java
interface Days {
    final int MON = 1;
    final int TUE = 2;
    final int WED = 3;
    final int THU = 4;
    final int FRI = 5;
    final int SAT = 6;
    final int SUN = 7;
}
```

- Just like methods specifications, the final variables are implicitly public in an interface
Accessing Days

• final variables of the interface act like public instance variables

```java
class Calendar implements Days {
    ...
}
```

• Can access days via a Calendar object

```java
Calendar c = new Calendar();
System.out.println(c.MON + c.TUE);
```
It’s Exercise Time