UNIVERSITY OF CALGARY

Department of Computer Science

Java Review

Object Oriented Programming in Java

Contents

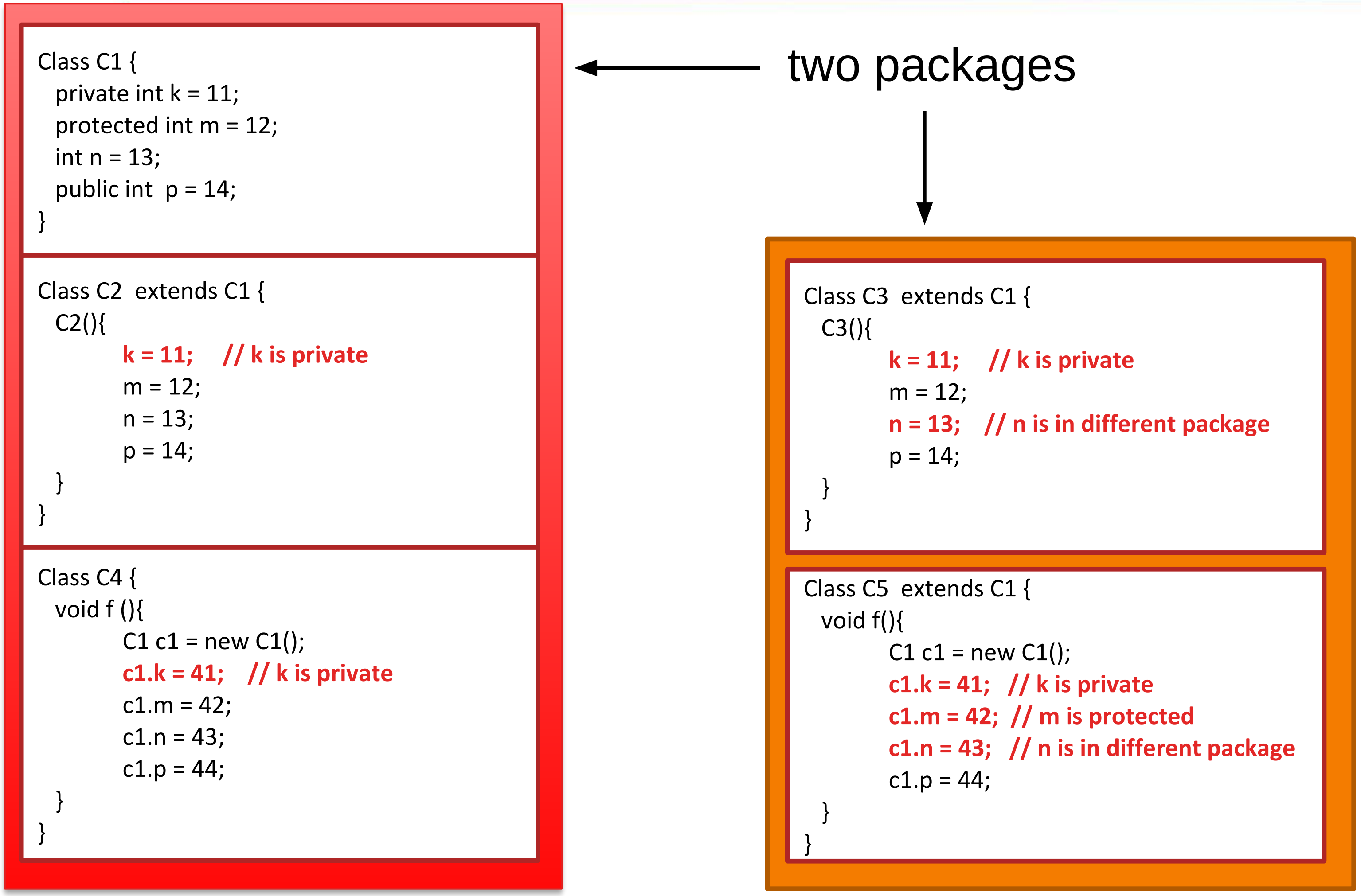
Accessibility control

- The private modifier indicates methods and fields that can be used only by this class
- The protected modifier means that a method or a data field is accessible to derived classes and in the package that includes the class that declares the method or the data field.
- A default modifier is no modifier at all, which indicates access to methods and fields in the package that includes the class that declares the methods or the data fields.
- Methods and fields declared public can be used by any other object.

Accessibility control

place of access	**`private`**	**`protected`**	no modifier	**`public`**
same class	yes	yes	yes	yes
same package subclass	no	yes	yes	yes
same package non-subclass	no	yes	yes	yes
different package subclass	no	yes	no	yes
different package non-subclass	no	no	no	yes

Accessibility control



two packages

```
Class C1 {
  private int k = 11;
  protected int m = 12;
  int n = 13;
  public int  p = 14;
}
```

```
Class C2  extends C1 {
  C2(){
        k = 11;     // k is private
        m = 12;
        n = 13;
        p = 14;
  }
}
```

```
Class C4 {
  void f (){
        C1 c1 = new C1();
        c1.k = 41;    // k is private
        c1.m = 42;
        c1.n = 43;
        c1.p = 44;
  }
}
```

```
Class C3  extends C1 {
  C3(){
        k = 11;     // k is private
        m = 12;
        n = 13;    // n is in different package
        p = 14;
  }
}
```

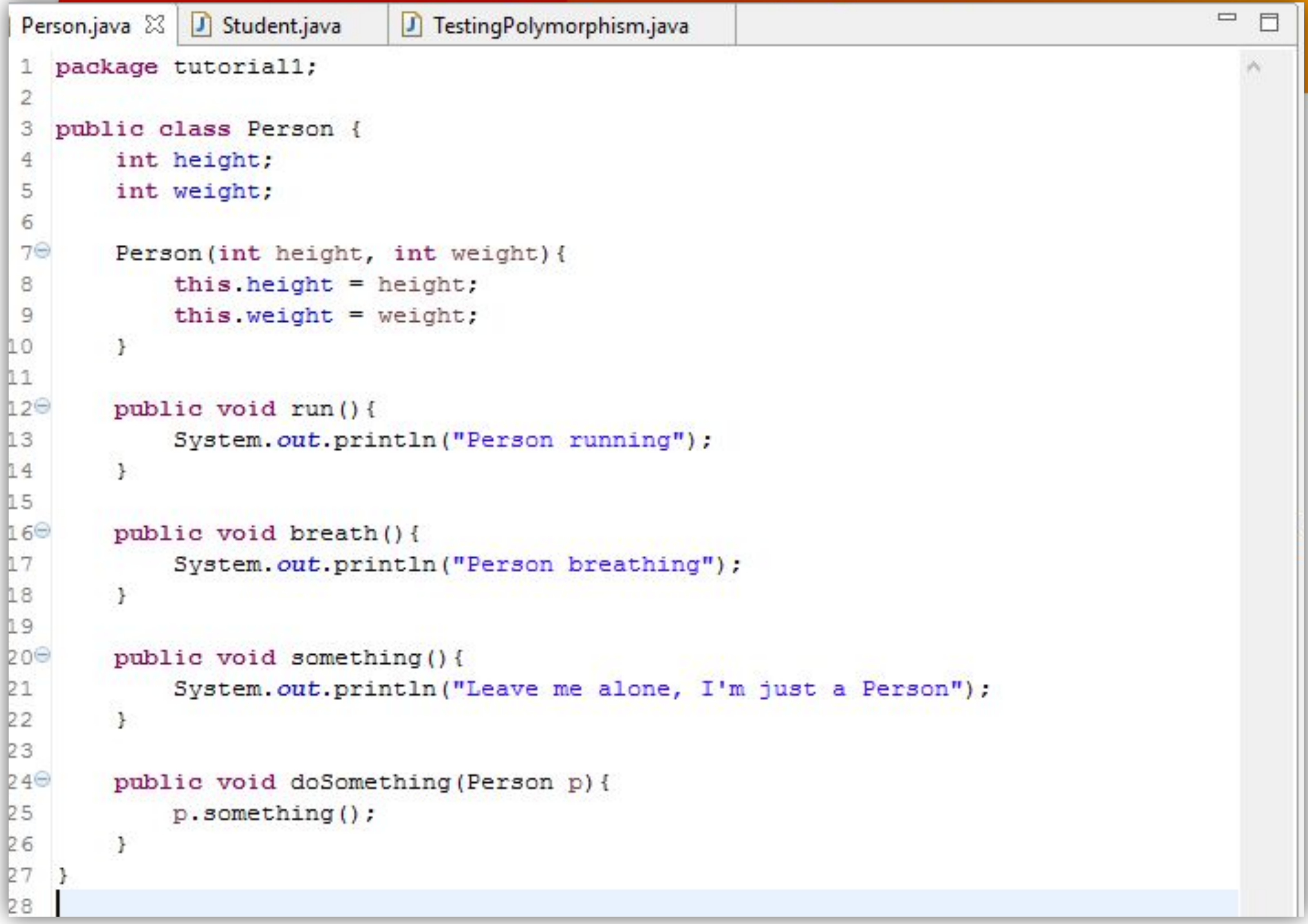
```
Class C5  extends C1 {
  void f(){
        C1 c1 = new C1();
        c1.k = 41;   // k is private
        c1.m = 42;  // m is protected
        c1.n = 43;   // n is in different package
        c1.p = 44;
  }
}
```

Polymorphism

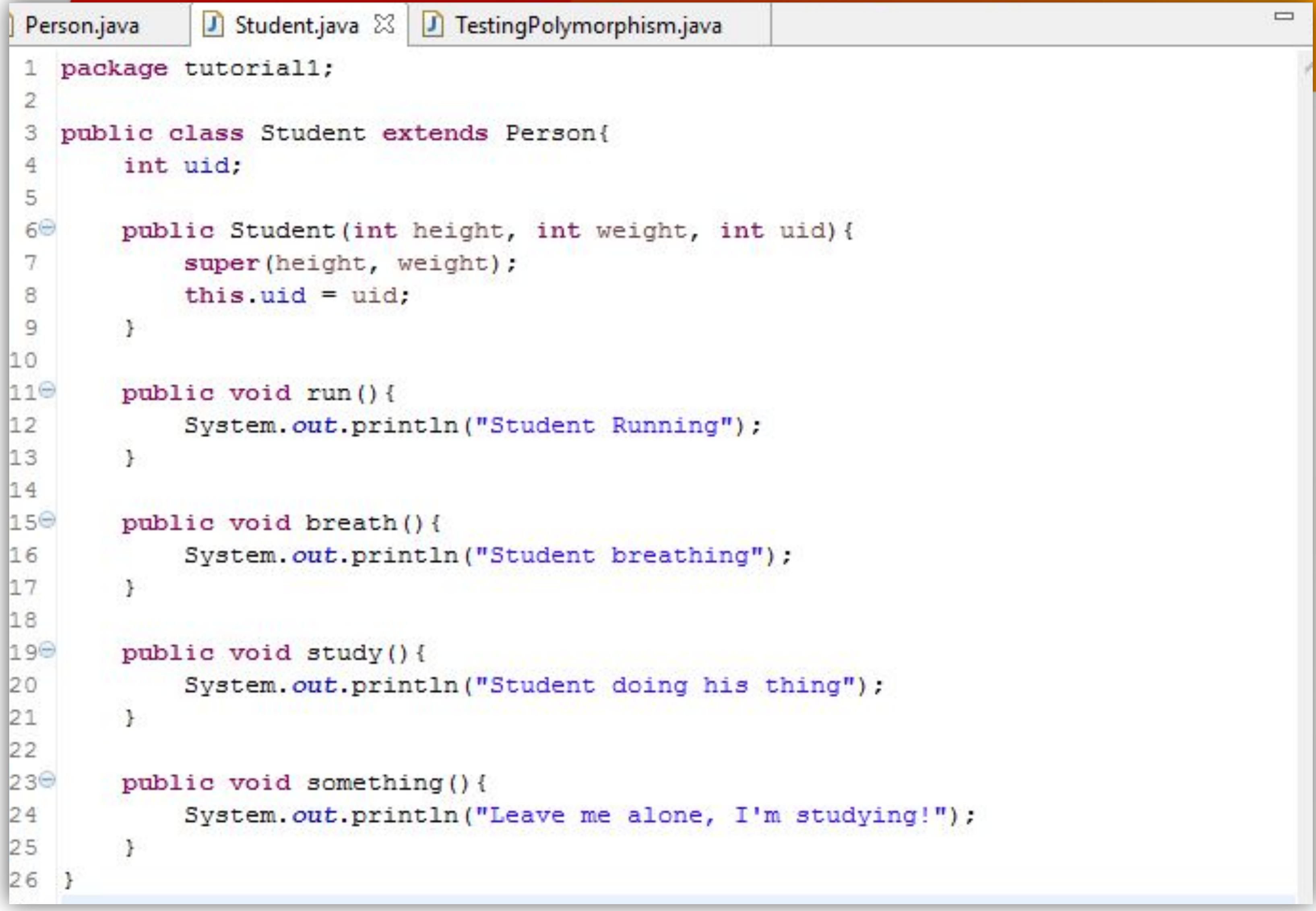
Java checks the type of object to which a reference is made and chooses the method appropriate for this type:

- at compilation time producing static binding
- during run time producing dynamic binding.

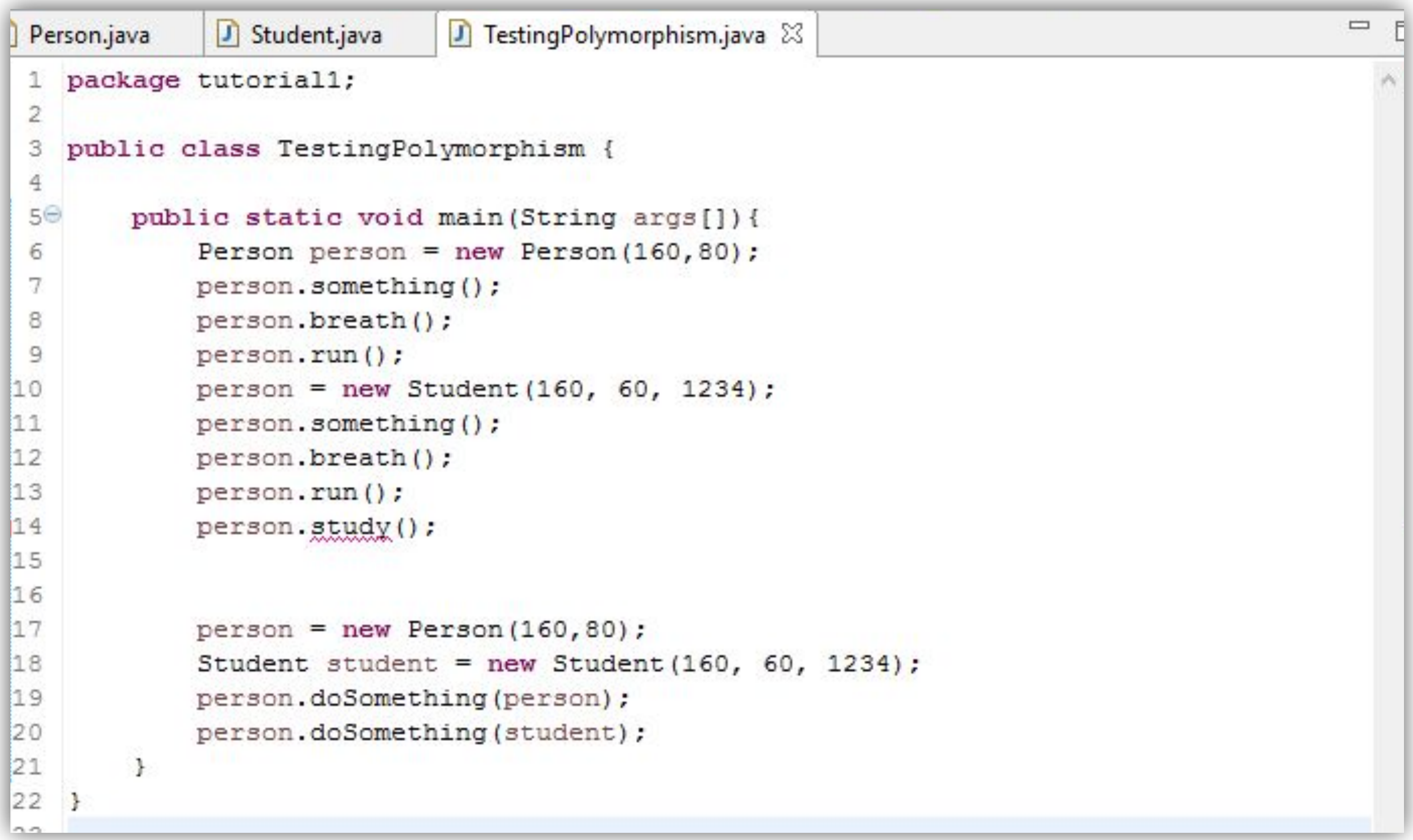
Polymorphism is an ability to associate with the same method name different meanings through the mechanism of dynamic binding.



```java
package tutorial1;

public class Person {
    int height;
    int weight;

    Person(int height, int weight){
        this.height = height;
        this.weight = weight;
    }

    public void run(){
        System.out.println("Person running");
    }

    public void breath(){
        System.out.println("Person breathing");
    }

    public void something(){
        System.out.println("Leave me alone, I'm just a Person");
    }

    public void doSomething(Person p){
        p.something();
    }
}
```



```java
package tutorial1;

public class Student extends Person{
    int uid;

    public Student(int height, int weight, int uid){
        super(height, weight);
        this.uid = uid;
    }

    public void run(){
        System.out.println("Student Running");
    }

    public void breath(){
        System.out.println("Student breathing");
    }

    public void study(){
        System.out.println("Student doing his thing");
    }

    public void something(){
        System.out.println("Leave me alone, I'm studying!");
    }
}
```

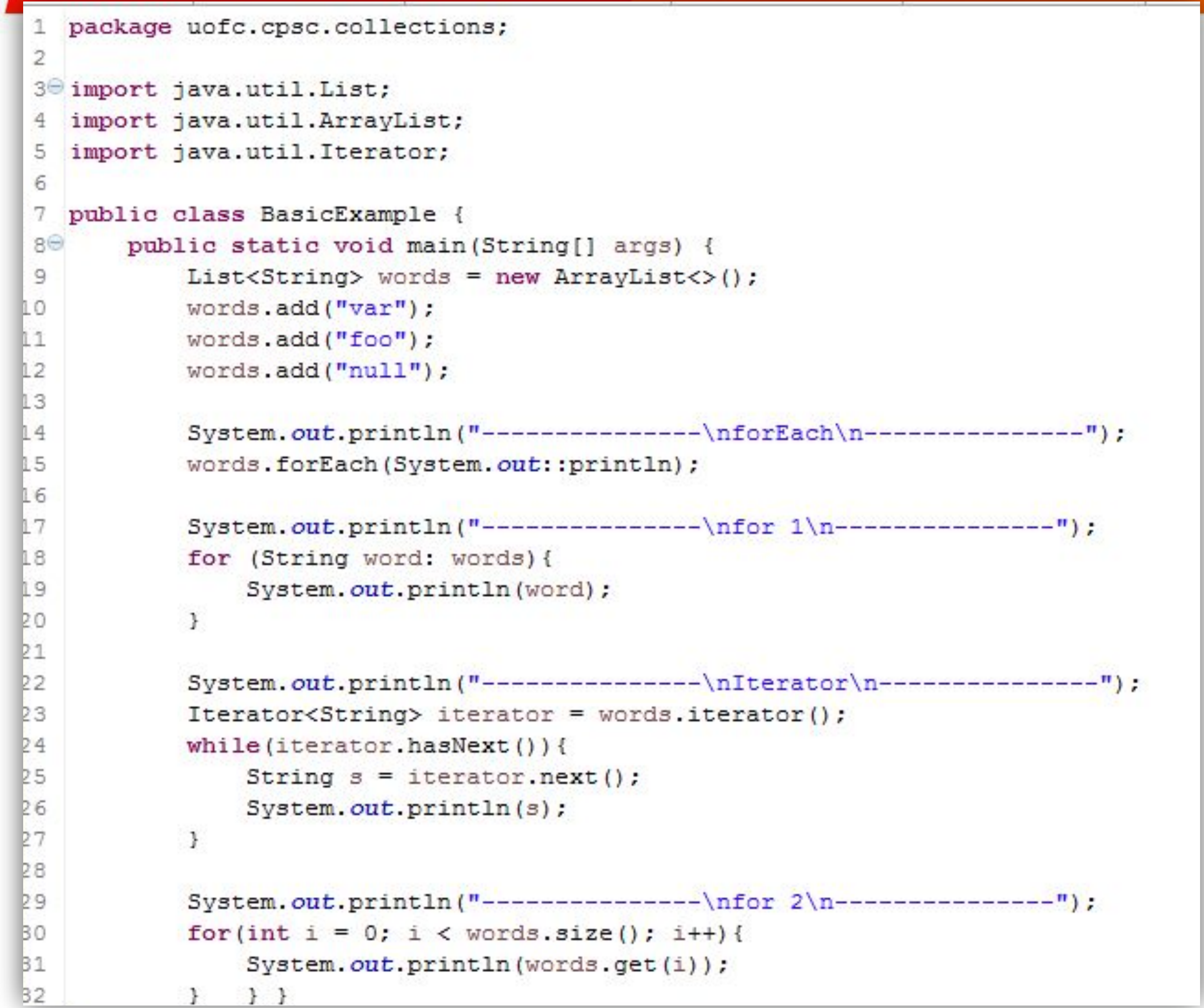


```java
package tutorial1;

public class TestingPolymorphism {

    public static void main(String args[]){
        Person person = new Person(160,80);
        person.something();
        person.breath();
        person.run();
        person = new Student(160, 60, 1234);
        person.something();
        person.breath();
        person.run();
        person.study();


        person = new Person(160,80);
        Student student = new Student(160, 60, 1234);
        person.doSomething(person);
        person.doSomething(student);
    }
}
```

A small peek to collections

- A *collection* is a data structure which contains and processes a set of data.
- Access to the stored data is only possible via predefined methods.
  - List
    - Array List
    - Linked List
  - Map
    - HashMap

A small peek into collections



```java
package uofc.cpsc.collections;

import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class BasicExample {
    public static void main(String[] args) {
        List<String> words = new ArrayList<>();
        words.add("var");
        words.add("foo");
        words.add("null");

        System.out.println("---------------\nforEach\n---------------");
        words.forEach(System.out::println);

        System.out.println("---------------\nfor 1\n---------------");
        for (String word: words){
            System.out.println(word);
        }

        System.out.println("---------------\nIterator\n---------------");
        Iterator<String> iterator = words.iterator();
        while(iterator.hasNext()){
            String s = iterator.next();
            System.out.println(s);
        }

        System.out.println("---------------\nfor 2\n---------------");
        for(int i = 0; i < words.size(); i++){
            System.out.println(words.get(i));
        }   } }
```

Practice

- Add a new method to the class **Employee** named **doYourJob** that prints “I’m on it Boss”.
- Modify the class Manager in order to override the methods defined in Employee including the method **doYourJob**, make sure that the method **toString** prints the right values for the manager and the method **doYourJob** prints “Every one get to work”. Remove all the manager specific methods.
- Do the same for the class **Secretary**
- Finally, use the class **testInheritance** to create two instances of each class (Employee, Manager and Secretary). Store all the objects in an ArrayList and call their methods **toString** and **doYourJob** inside of a loop.