CSE 143
Lecture 3

Inheritance

slides created by Marty Stepp

http://www.cs.washington.edu/143/

More `ArrayIntList`

- Let's add some new features to our `ArrayIntList` class:
  1. A method that allows client programs to print a list's elements
  2. A constructor that accepts an initial capacity

  (By writing these we will recall some features of objects in Java.)

- Printing lists: You may be tempted to write a `print` method:

  ```
  // client code
  ArrayIntList list = new ArrayIntList();
  ...
  list.print();
  ```

  – Why is this a bad idea? What would be better?

The toString method

- Tells Java how to convert an object into a `String`

```
ArrayIntList list = new ArrayIntList();
System.out.println("list is " + list);
           // ("list is " + list.toString());
```

- Syntax:

```
public String toString() {
    code that returns a suitable String;
}
```

- Every class has a `toString`, even if it isn't in your code.
  - The default is the class's name and a hex (base-16) number:

```
ArrayIntList@9e8c34
```

toString solution

```
// Returns a String representation of the list.
public String toString() {
    if (size == 0) {
        return "[]";
    } else {
        String result = "[" + elementData[0];
        for (int i = 1; i < size; i++) {
            result += ", " + elementData[i];
        }
        result += "]";
        return result;
    }
}
```

Multiple constructors

- existing constructor:

```
public ArrayIntList() {
    elementData = new int[1000];
    size = 0;
}
```

- Add a new constructor that accepts a capacity parameter:

```
public ArrayIntList(int capacity) {
    elementData = new int[capacity];
    size = 0;
}
```

  - The constructors are very similar. Can we avoid redundancy?

`this` keyword

- **`this`** : A reference to the *implicit parameter*
  (the object on which a method/constructor is called)

- Syntax:
  - To refer to a field: `this.`**field**
  - To call a method: `this.`**method**`(`**parameters**`);`
  - To call a constructor from another constructor: `this(`**parameters**`);`

Revised constructors

```
public ArrayIntList(int capacity) {
    elementData = new int[capacity];
    size = 0;
}


public ArrayIntList() {
    this(1000);   // calls other constructor
}
```

Exercise

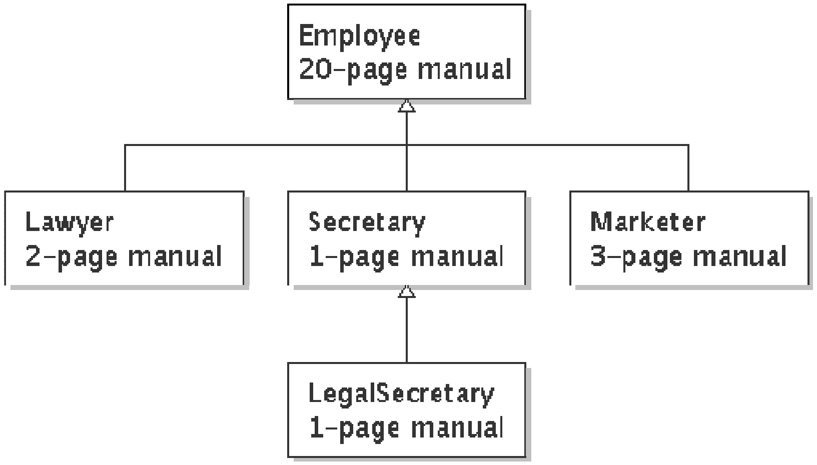
- Write a class called `StutterIntList`.
  - Its constructor accepts an integer *stretch* parameter.
  - Every time an integer is added, the list will actually add *stretch* number of copies of that integer.

- Example usage:

```
StutterIntList list = new StutterIntList(3);
list.add(7);        // [7, 7, 7]
list.add(-1);       // [7, 7, 7, -1, -1, -1]
list.add(2, 5);     // [7, 7, 5, 5, 5, 7, -1, -1, -1]
list.remove(4);     // [7, 7, 5, 5, 7, -1, -1, -1]
System.out.println(list.getStretch());   // 3
```

Inheritance

- **inheritance**: Forming new classes based on existing ones.
  - a way to share/**reuse code** between two or more classes
  - **superclass**: Parent class being extended.
  - **subclass**: Child class that inherits behavior from superclass.
    - gets a copy of every field and method from superclass
  - **is-a relationship**: Each object of the subclass also "is a(n)" object of the superclass and can be treated as one.



Inheritance syntax

```
public class name extends superclass {
```

- Example:

```
public class Lawyer extends Employee {
    ...
}
```

- By extending `Employee`, each `Lawyer` object now:
  - receives a copy of each method from `Employee` automatically
  - can be treated as an `Employee` by client code

Overriding methods

- **override**: To replace a superclass's method by writing a new version of that method in a subclass.
  - No special syntax is required to override a method.
    Just write a new version of it in the subclass.

```
public class Lawyer extends Employee {
    // overrides getSalary method in Employee class;
    // give Lawyers a $5K raise
    public double getSalary() {
        return 55000.00;
    }
}
```

super keyword

- Subclasses can call overridden methods with `super`

  `super.`**method**`(`**parameters**`)`

  - Example:

```
public class Lawyer extends Employee {
    // give Lawyers a $5K raise (better)
    public double getSalary() {
        double baseSalary = super.getSalary();
        return baseSalary + 5000.00;
    }
}
```

  - This version makes sure that Lawyers always make $5K more than Employees, even if the Employee's salary changes.

Calling super constructor

`super(`**parameters**`);`

- Example:

```
public class Lawyer extends Employee {
    public Lawyer(int years) {
        super(years);  // calls Employee constructor
    }
    ...
}
```

- The `super` call must be the first statement in the constructor.
- Constructors are not inherited; If you extend a class, you must write all the constructors you want your subclass to have.

Exercise solution

```
public class StutterIntList extends ArrayIntList {
    private int stretch;

    public StutterIntList(int stretchFactor) {
        super();
        stretch = stretchFactor;
    }

    public StutterIntList(int stretchFactor, int capacity) {
        super(capacity);
        stretch = stretchFactor;
    }

    public void add(int value) {
        for (int i = 1; i <= stretch; i++) {
            super.add(value);
        }
    }

    public void add(int index, int value) {
        for (int i = 1; i <= stretch; i++) {
            super.add(index, value);
        }
    }

    public int getStretch() {
        return stretch;
    }
}
```