Extending Classes

The `extends` keyword allows you to define a new class based on an existing class. This way, you can define new versions of classes without having to copy and paste their source code.

Manager: _____   Recorder: _____

Presenter: _____   Reflector: _____

Content Learning Objectives

After completing this activity, students should be able to:

- Explain what it means for one class to extend another.
- Summarize uses of the keywords `extends` and `super`.
- Write a new method for an existing Java library class.

Process Skill Goals

During the activity, students should make progress toward:

- Making conclusions based on IDE hints and program output. (Critical Thinking)

Facilitation Notes

This activity requires some prior knowledge of BigInteger and the `String.replace()` method. All that's needed is a 10-minute mini lecture (e.g., using JShell) that shows how to create and add two BigIntegers, and how to find and replace strings.

BigInteger was originally chosen for this activity, because (1) it's a useful class in the Java library, and (2) it's not declared as `final` (so it can be extended). The new feature this activity is adding to BigInteger is the ability to work with comma separators.

Highlight important aspects of the documentation for BigInteger. For example, mention that BigInteger extends Number, which extends Object. Notice the static constant fields. In addition to the constructors, there is a static `valueOf` method.

Prevent students from spending too much time on **#6** and **#7**. You can tell them the number of methods; they should be able to count the fields and constructors. Introduce "is a" and "has a" terminology when reporting out.

Key questions: **#5**, **#8**, **#16**

Source files: *MyBigInt.java*

Model 1   My Big Integer

The following class extends the functionality of `BigInteger` to allow comma-separated strings
(e.g., `"123,465,789"`). The UML diagram summarizes the relationship between the two classes.

```java
1   import java.math.BigInteger;
2
3   public class MyBigInt extends BigInteger {
4
5       public MyBigInt(String val) {
6           // remove comma characters
7           super(val.replace(",",  ""));
8       }
9
10      public String toString() {
11          // start with the decimal representation
12          String str = super.toString();
13          StringBuilder sb = new StringBuilder(str);
14
15          // insert comma separators every three digits
16          for (int i = sb.length() - 3; i > 0; i -= 3) {
17              sb.insert(i, ',');
18          }
19          return sb.toString();
20      }
21
22  }
```

| BigInteger | *superclass* |

| MyBigInt | *subclass* |
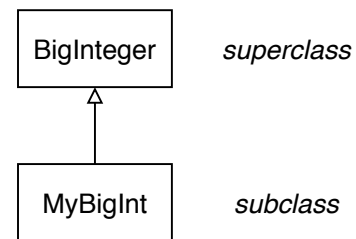
Questions (20 min)   Start time:

1. Based on the UML diagram:

a) Which class is the subclass? MyBigInt

b) Which class is the superclass? BigInteger

2. The keyword `super` behaves like the keyword `this`, except that it refers to the superclass.
On the following lines, which method (in which class) is being invoked?

a) Line 7: constructor in BigInteger

b) Line 11: toString in BigInteger

c) Line 18: toString in StringBuilder

3. Open *MyBigInt.java* in your editor. Copy the following code snippets into the `main` method, one at a time (without the others), and run them. Record the results in the table below.

Java Code	Result
`BigInteger bi = new BigInteger("123456789");` `System.out.println(bi);`	123456789
`MyBigInt bi = new MyBigInt("123456789");` `System.out.println(bi);`	123,456,789
`BigInteger bi = new BigInteger("123,456,789");` `System.out.println(bi);`	NumberFormatException
`MyBigInt bi = new MyBigInt("123,456,789");` `System.out.println(bi);`	123,456,789

`BigInteger bi1 = new BigInteger("123456789");`
`MyBigInt bi2 = new MyBigInt("123,456,789");`
`System.out.println(bi1.equals(bi2));`
`System.out.println(bi2.equals(bi1));`
	true
	
	true

4. Based on the results of the previous question, summarize what the source code for each method does:

a) `MyBigInt` constructor

It first removes any commas from the given string. It then invokes the constructor of `BigInteger` to initialize the object.

b) `MyBigInt.toString`

It first invokes the `toString` method of `BigInteger`. It then uses a `StringBuilder` to insert commas into the result.

c) `MyBigInt.equals`

It compares the numerical contents of one `BigInteger` with another. (The commas in `MyBigInt` are not stored; they are inserted by `toString`.)

5. Why do you think `bi2.equals(bi1)` compiles and runs correctly, even though the `MyBigInt` class does not define an `equals` method?

`MyBigInt` uses (inherits) the `equals` method from `BigInteger`.

6. Refer to the documentation for BigInteger and the source code for `MyBigInt`. How many public items are defined in each class?

a) `BigInteger` fields: 4

b) `BigInteger` constructors: 8

c) `BigInteger` methods: 50

d) `MyBigInt` fields: 0

e) `MyBigInt` constructors: 1

f) `MyBigInt` methods: 1

7. Answer each question by typing the following code in `main` and pressing Ctrl+Space to list possible completions.

a) How many public fields does a `MyBigInt` have?   `bi2.`
   4   (*Hint:* scroll down to the bottom)

b) How many constructors does a `MyBigInt` have?   `bi2 = new MyBigInt(`
   1   (ignore anonymous inner types)

c) About how many methods does a `MyBigInt` have?   `bi2.`
   61   (not counting the `main` method)

8. Notice that `MyBigInt` has most of the same fields and methods as `BigInteger`. Non-private fields and methods are ***inherited*** when extending a class. Based on your answers to the previous two questions, what is <u>not</u> inherited? Explain your reasoning.

Constructors are not inherited. `BigInteger` has 8 constructors, but `MyBigInt` has only 1.

9. Make the following changes to *MyBigInt.java*, and summarize the compiler errors.

a) Rewrite the constructor using two lines of code:

```
String str = val.replace(",", "");
super(str);
```

Constructor call must be the first statement in a constructor.

b) Remove all code from the body of the constructor.

Implicit super constructor BigInteger() is undefined.
Must explicitly invoke another constructor.

c) Remove the constructor altogether.

Implicit super constructor BigInteger() is undefined for default constructor.
Must define an explicit constructor.

Model 2   Adding New Methods

Add the following method to the `MyBigInt` class:

```java
public MyBigInt reverse() {
    String str = super.toString();
    final int N = str.length();

    // reverse the digits in the string
    StringBuilder sb = new StringBuilder(N);
    for (int i = 0; i < N; i++) {
        int j = N - 1 - i;
        sb.append(str.charAt(j));
    }
    return new MyBigInt(sb.toString());
}
```

Add the following code to the `main` method:

```java
BigInteger bi1 = new BigInteger("12345678");
MyBigInt bi2 = new MyBigInt("12,345,678");
System.out.println(bi1.reverse());
System.out.println(bi2.reverse());
```

Questions (20 min)   Start time:

10. Attempt to compile and run the program. Explain the error in `main`.

The method `reverse()` is undefined for the type `BigInteger`. It was defined only in the `MyBigInt` class.

11. Remove the line that caused the error, and run the program. What is the result?

87,654,321

12. Which `toString` method (in which class) is invoked on the first line of `reverse`?

`BigInteger.toString()`

13. Explain why `reverse()` does not need to worry about the placement of commas.

The string returned from `BigInteger.toString()` does not contain any commas. The commas are only added when `System.out.println` calls `MyBigInt.toString()`.

14. Consider a method `isPalindrome()` that determines whether a `MyBigInt` has the same digits forward and backward. For example, 123,321 and 12,321 are palindromes, but 123,421 and 12,341 are not. How could you implement this method using one line of code?

```java
public boolean isPalindrome() {
    return this.equals(this.reverse());   // note: "this." is optional
}
```

15. Why is the one-line implementation inefficient, especially for very large integers?

It converts the integer to a string, then it makes a copy of the string, and then it copies the string into a new `MyBigInt`. The two extra copies are unnecessary just for checking digits.

16. Rewrite `isPalindrome()` to be more efficient. (*Hint:* Use the source code of `reverse()` as a starting point.)

```java
public boolean isPalindrome() {

    String str = super.toString();
    final int N = str.length();

    // check each pair of digits
    for (int i = 0; i < N / 2; i++) {
        int j = N - 1 - i;
        if (str.charAt(i) != str.charAt(j)) {
            return false;
        }
    }
    return true;

}
```

17. Add your solution to *MyBigInt.java*, and make sure it works. What code can you add to `main` to test the `isPalindrome` method?

```java
System.out.println(new MyBigInt("123321").isPalindrome());
System.out.println(new MyBigInt("12321").isPalindrome());
System.out.println(new MyBigInt("123421").isPalindrome());
System.out.println(new MyBigInt("12341").isPalindrome());
```