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.

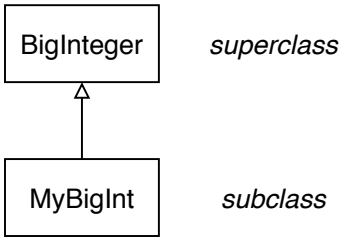
Process Skill Goals

During the activity, students should make progress toward:

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

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.




```
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 }
```

Questions (20 min) Start time:

1. Based on the UML diagram:

a) Which class is the subclass?

b) Which class is the superclass?

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:

b) Line 11:

c) Line 18:

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

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

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

a) `MyBigInt` constructor

b) `MyBigInt.toString`

c) `MyBigInt.equals`

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

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:

b) `BigInteger` constructors:

c) `BigInteger` methods:

d) `MyBigInt` fields:

e) `MyBigInt` constructors:

f) `MyBigInt` methods:

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.`
(*Hint:* scroll down to the bottom)

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

c) About how many methods does a `MyBigInt` have? `bi2.`
(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 not inherited? Explain your reasoning.

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);
```

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

c) Remove the constructor altogether.

Model 2 Adding New Methods

Add the following method to the `MyBigInt` class:

```
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:

```
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`.

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

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

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

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?

```
public boolean isPalindrome() {



}
```

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

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

```
public boolean isPalindrome() {



}
```

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