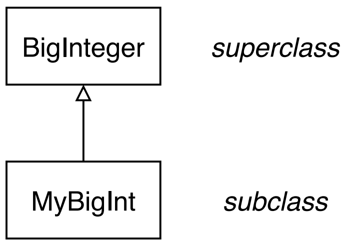
Model 1 My Big Integer

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? 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 not 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.