Review of Java

- Higher Level Language Concepts
  - Names and Reserved Words
  - Expressions and Precedence of Operators
  - Flow of Control – Selection
  - Flow of Control – Repetition
  - Arrays

Slides derived from Professor Bob Wilson (UMass Boston)

Review of Java

- This is a review of the material you should have learned in CS110 or CSIT 114/115
- If you are not completely familiar with this material, please see me. You should not be taking this course!
- Don’t expect that you’ll catch up as we go through the CS210 material. That’s futile!

Names and Reserved Words

- Names are used for classes and methods:
  - Classes: Scanner, BingoBall
  - Methods: next( ), hasNext( )
- Names are used for variables and constants
  - Variables: i, j, current, currentValue, isOn
  - Constants: SIZE, MAX_VALUE
- Reserved words are java language identifiers:
  - Examples: class, return, public, static, int, boolean

Expressions and Precedence

- An expression is an ordered sequence of:
  - Operators: +, -, *, /, (type), ., [ ], ++, ==
  - Operands: variables and/or constants
- The precedence of operators determines the order of evaluation for expressions:
  - Highest: [ ], . ( ), ++, --
  - Next: +(unary), -(unary), ~, !
  - Next: new, (type)
  - Next: *, /, %
  - Next: +(binary), -(binary)
  - Etc.

Flow of Control - Selection

- If statements with optional else clauses:

```
if (boolean condition)
   statement;
else
   statement;
```

- Switch statements

```
switch (integer value) {
   case FIRST_VALUE:
        statements;
   case SECOND_VALUE:
        statements;
   default:
        statements;
}
```

Flow of Control - Repetition

- While

```
while (scan.hasNext()) {
   statements;  // repeated until false above
}
```

- For

```
for (int i = 0; i < MAX; i++) {
   statements;  // repeated until false above
}
```

- Do … while

```
do {
   statements;  // repeated until false below
} while (!done);
```

Arrays

- Arrays are a group of elements that can be referenced via a name and an index value
- Declaring an array with or w/o initialization

```
int [] digits = new int [10];
```

OR

```
int [] digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

- Setting the value of an element of an array

```
digits [0] = 0;
```

- Using the values of elements of an array

```
int sum = digits[2] + digits[4];
```

Review of Java

- Object Oriented Programming Concepts
  - Objects and Classes
  - Encapsulation, Constructors, and Methods
  - References and Aliases
  - Interfaces and Inheritance
  - Class Hierarchies and Polymorphism
  - Generic Types (ArrayList Class)
  - Exceptions

Objects and Classes

- Class Definition

```
public class ClassName
{
    // attributes

    // methods
}
```

- Instantiating Objects using Classes

```
ClassName myClassName = new ClassName();
```

Encapsulation

- Encapsulation of Attributes

```
public class ClassName
{
   // constants
   public static final int MAX_SIZE = 20;
   private static final int DEFAULT_SIZE = 10;

   // class variables
   private static int largestSizeOfAll;

   // instance variables
   private int mySize;
}
```

Constructors and Methods

- Constructor (ClassName with no return type)

```
public class ClassName
{
    public ClassName (parameter list if any)
    {
        statements;
    }
```

- Method (Method name with a return type)

```
    public type methodName(parameter list if any)
    {
        statements;
    }
}
```

Using References

- Using a class constant

```
if (size <= ClassName.MAX_SIZE)
    statement;
```

- Using a class method

```
type returnValue = ClassName.methodName(...);
```

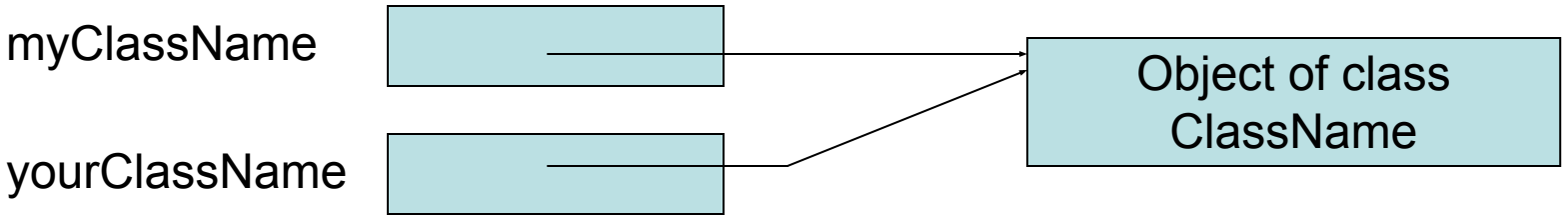
- Using an instance method via a reference

```
type returnValue = myClassName.methodName(...);
```

Aliases and Garbage Collection

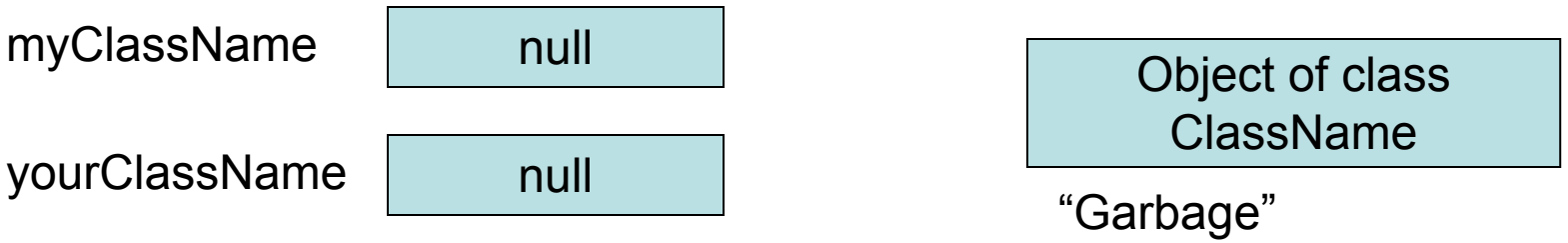
- Creating an alias of a reference

```
ClassName yourClassName = myClassName;
```



- Making object eligible for garbage collection

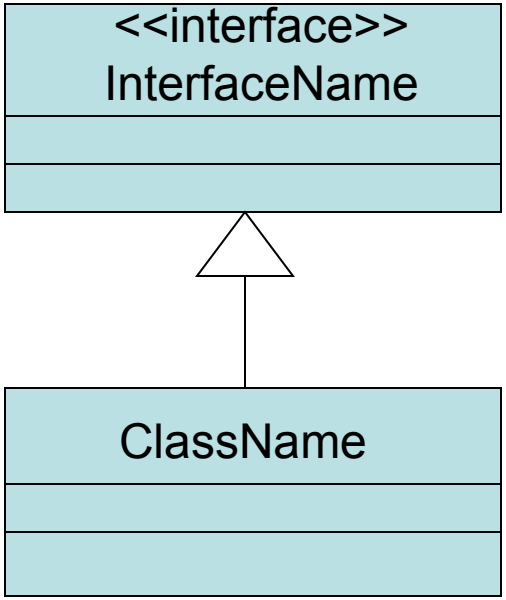
```
myClassName = yourClassName = null;
```



Interfaces

- A class that implements an interface

```
public class ClassName implements InterfaceName
{
    . . .
}
```



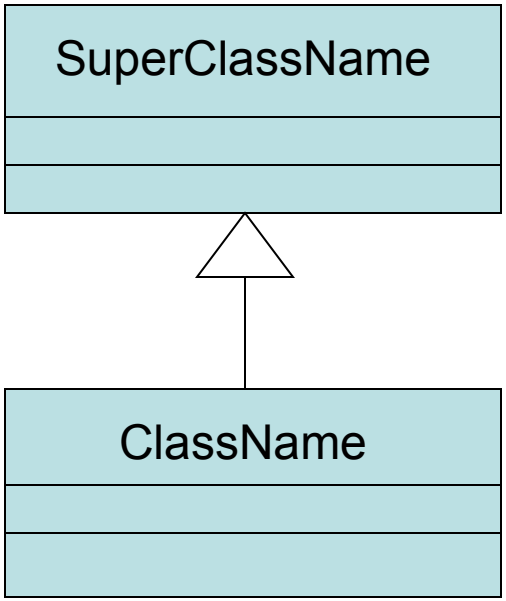
Interface defines the method signature for all required methods

Class must define code for all methods defined in InterfaceName

Inheritance

- A class that extends another class

```
public class ClassName extends SuperClassName
{
    . . .
}
```



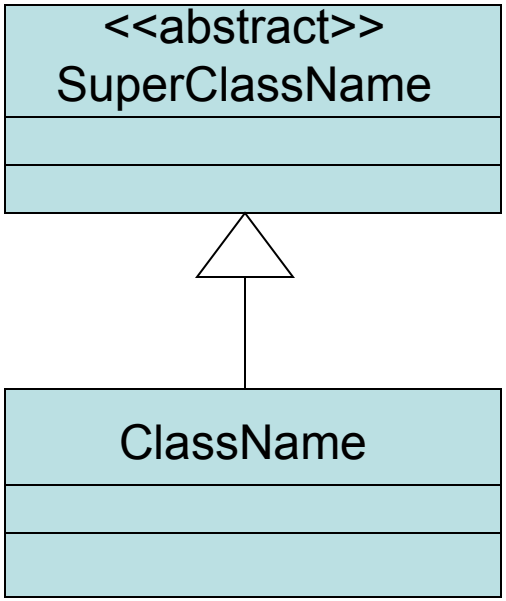
SuperClass defines all the methods for all SubClasses

SubClass may define code to override some or all of the SuperClass methods

Inheritance

- A class that extends an abstract class

```
public class ClassName extends SuperClassName
{
    . . .
}
```



SuperClass defines some of the methods for all SubClasses, but only defines method signatures for the rest of the methods

SubClass may define code to override some or all of the SuperClass methods, but must define code for the signatures

Class Hierarchies and Polymorphism

- An object reference variable may hold a reference to any compatible type of object
- Compatibility may be via implementing an interface or inheritance from another class

```
ClassName a = new ClassName();

InterfaceName b = new ClassName();

SuperClassName c = new ClassName();
```

- Object behaves as class it was “born as” (i.e. class used with the new operator)

Generic Types

- Collection classes like the ArrayList class can be defined to hold a specific type of object via a generic type designation <T>

```
ArrayList<String> myList = new ArrayList<String>();
```

- We will use generics often in CS210 with many other types of “collection” classes

Exceptions

- When code encounters a situation that is impossible for it to resolve, it may throw an Exception object, e.g. NameOfException instead of executing its normal return
- If a method may throw an exception, it should indicate that in its method header

```
public void methodName() throws NameOfException
{
    if (boolean condition of impossible situation)
        throw new NameOfException();
}
```

Exception Handling

- Code that calls a method that may throw a checked exception must use `try-catch` or indicate that it `throws` that exception in its own method header

```
try
{
    statements with call to methodName();
}
catch (NameOfException e)  // may be multiple catch clauses
{
    statements to recover from occurrence of exception
}
finally  // optional finally clause
{
  statements always performed, e.g. clean up actions
}
```

File Input: Example

```
import java.util.Scanner;
import java.io.*;

public class FileDisplay
{
  public static void main (String [] args)
                  throws IOException
  {
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter name of file to display");
    File file = new File(scan.nextLine());

    Scanner fileScan = new Scanner (file);
    while (fileScan.hasNext())
      System.out.println(fileScan.nextLine());
  }
}
```

File Output: Example

```
import java.util.Scanner;
import java.io.*;

public class FileWrite
{
  public static void main (String [] args) throws IOException
  {
    // Get filename and instantiate File object as before

    PrintStream out = new PrintStream(file);
    while (scan.hasNext()) {
      String line = scan.nextLine();
      if (line.equals("END"))     // A sentinel String value
         break;
      else
         out.println(line);
    }
    out.close();
  }
}
```