Reusable Classes

- In general, instantiable classes should only have one task

- There are 4 general categories of "object tasks":

  - User Interface Objects
    - handles the interaction between the user and the program
    - often reusable (unless too specific of an interaction)
    - examples: InputBox, OutputBox, etc.

  - Controller Objects
    - supervises other objects in the program
    - rarely reusable (usually very specific to the program)
    - examples: "main" classes

  - Application Logic Objects
    - "real world" objects (can be an imaginary non-real-world thing also)
    - usually reusable
    - examples: Student, AddressBook, etc.

  - Storage Objects
    - an object that is used to store, sort, maintain data
    - often reusable (depends upon how specific the data must be)
    - examples: MyArray, Vector, LinkedList, Stack, Queue, etc.

Method Overloading

- Definition: having multiple methods of a class with the same name but different signatures
  - Signature: the name and parameter list (number & type of parameters) of a method
  - Prototype: the FULL method header including all modifiers (e.g., public, static, etc.), return-type, and signature
  - Cannot have two methods in the same class with the same signature

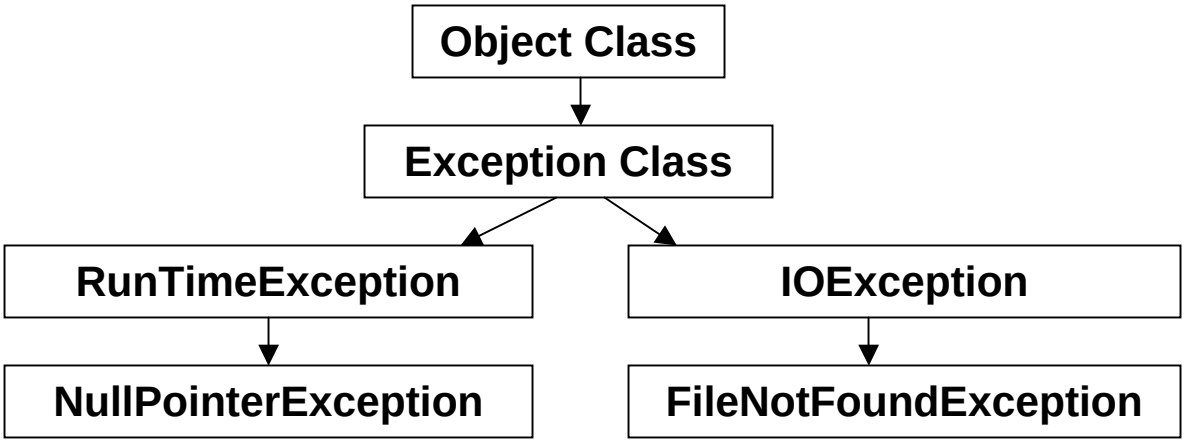
- Constructors are methods, and therefore can be overloaded
  - Having multiple constructors is a common way to increase the reusability of a class

```
class Menu {


    /* Data Members */

    private String header;
    private String[] choices;




    /* Constructors */

    public Menu (String header, int numChoices) {
        this.header = header;
        choices = new String[numChoices];
    }

    public Menu (int numChoices) {
        this.Menu (“Menu”, numChoices);
    }

    public Menu (){
        this.Menu (“Menu”, 2);
        this.setChoice (“Yes or True”, 0);
        this.setChoice (“No or False”, 1);
    }
```

```
    /* Methods */


    public void setChoice (String s, int index)
        throws ArrayIndexOutOfBoundsException {
        choices[index] = s;
    }


    public int getChoice () throws IOException {
        BufferedReader stdin = new BufferedReader
            (new InputStreamReader (System.in));

        System.out.println (header);

        for (int i = 0; i < choices.length; i++) {
            System.out.println (i + ": " + choices[i]);
        }
        boolean valid = false;
        while (!valid) {
            System.out.print ("Enter Choice (0-" +
                                choices.length-1 + ") =>
                        ");
            try {
                int choice = Integer.parseInt
                            (stdin.readLine());
                if (choice > choices.length – 1)
                    System.out.println ("Invalid Entry");
                else
                    valid = true;
            }
            catch (NumberFormatException e) {
                System.out.println ("Invalid Entry");
            }
        }
        return choice;

    }

}
```

Inheritance

- Inheritance allows one to define one class in terms of another
  - The newly defined class is called the derived class
  - The class from which the new class is derived is called the base class or super class
  - Example: Exceptions (come from Exception class)



NOTE: ALL classes are ultimately derived from the Object class

- Why Inheritence?
  - models the real world through "...is a ..." relationships
    - e.g., a student is a person
    - e.g., a car is vehicle
  - code re-use
    - allows one to add functionality to existing classes

❖ The Derived Class

- Inherits (but not necessarily obtains access to) the following:
  - ALL Data Members of the base class
  - ALL methods of the base class except constructors
- Has access to ALL members of the base class that are declared as public or protected
- Does NOT have access to any private data members directly
- Declaration Syntax:

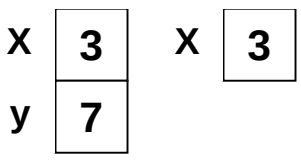
```
class <derived class' name> extends <base class' name>
```

- the name of the newly derived class must be different than the name its "parent"
- **extends** is a Java reserved word
- A class cannot extend multiple classes
- example:

```
class Base {
  private int x;
  public Base (int xx) {
       x = xx;
  }
}

class Derived extends Base {
  private int y;
  public Derived (int xx, int yy) {
       super (xx);
       y = yy;
  }
}
```

- The Java reserved word "super" is a reference to an object's super class. Only members of the super class can be accessed using the reference "super", including the constructor belonging to the base class

```
Derived der = new Derived (3, 7);
```



- References
  - A reference variable can store a reference to its class or any class derived from it
  - The constructor determines the actual type of the object it is referencing

```
Base b;      // can refer to a Base or a Derived
Derived d;   // can refer only to a Derived

b = new Base();       // b refers to a Base
b = new Derived();    // b refers to a Derived
d = new Derived();    // d refers to a Derived
d = new Base();       // ILLEGAL (d cannot refer to
                         a base)
```

  - When a Derived object is made, the Base's default constructor (no parameters) is automatically called UNLESS the Derived object's constructor implicitly calls a Base constructor.

  - If no "default" constructor exists for Base AND a Base constructor is not implicitly called by the Derived object's constructor, a syntax error will occur.

  - Can only type-cast an object that has been derived from another object

  - Example:

```
String s = "hello";
Vector v = new Vector (...);

v.insertElement (s);      // OK
Object o = new String();      // OK
s = v.elementAt(0);       // NOT OK
s = (String) v.elementAt(0);      // OK
o = v.elementAt(0);       // OK
```

❖ Overriding Methods

- If a method in a derived class has the same signature of a method in the base class, the base class' method is said to be overridden
- Overriding methods is one of the most common things done with derived classes
- Methods are overridden to modify the functionality of the method so that it has the correct behavior for the derived class
- An overridden method can call the base class' version of the method using "super"
- Example

```
class Base {
    private int x;
    public Base (int xx) {
        x = xx;
    }
    public String toString () {
        return "x = " + x;
    }
}
class Derived extends Base {
    private int y;
    public Derived (int xx, int yy) {
        super (xx);
        y = yy;
    }
    public String toString () {
        return super.toString() + ", y = " + y;
    }
}
```

- static methods cannot be overridden
- to prevent a method from being overridden, use the keyword **`final`**. Syntax:

```
public final <return-type> <method name> (<parameters>)
```

Polymorphism

- A reference variable can store references to types derived from it. e.g. an Object reference variable can store references to all objects since all classes are ultimately derived from the Object class.
- Since the derived class has all of the base class' members, these members can always be accessed from the base class
- When a method is called, Java calls the overridden version of the method that corresponds to the type of the object, NOT the object reference
- Example:

```
class Inherited {
    public static void main (String[] args) {
        Derived d = new Derived (1, 2);
        Base b1 = new Base (3);
        Base b2 = new Derived (4, 5);

        System.out.println ("d is " + d);
        System.out.println ("b1 is " + b1);
        System.out.println ("b2 is " + b2);
    }
}
```

output:
d is x = 1, y = 2
b1 is x = 3
b2 is x = 4, y = 5

Abstract Classes

- When there is no need to create instances of a super class, it is often declared as “abstract”
  - No instances (objects) can be created from an abstract class
  - Syntax:

```
abstract class <class name> { ... }
```

- An abstract class may contain abstract methods
  - abstract methods are not implemented (i.e., no method body); rather, contains just a method header ending with a semi-colon
  - Syntax:

```
abstract <modifiers> <return-type> <name>
(<params>);
```

  - Example:

```
abstract public void computeGPA ();
```

  - subclasses of the abstract super class MUST define (i.e., implement) the abstract methods
  - private and static methods cannot be declared as abstract