Lecture4

- Misc topics
  - static, this, super, final
- Abstract classes
- Interfaces
- Packages

Static members

- Specific to class, not individual objects
- Common to all objects
- Can be used with data or functions.
- E.g. main function is static

```
class staticExample
{
    staticExample( ) { }
    static int static_var = 1;
    static void static_fn( ) { }
    public static void main ( String args[ ] )
    {
      System.out.println (static_var); // No object is created
      static_fn( );                   // No object is created
   }
};
```

final in Java

- final can have several meanings in Java
  - final class cannot be extended
  - final methods cannot be overridden by members of child classes
  - final variables can only be assigned once

```
public final class myClass // Cannot be extended
    {
            public static final PI = 3.1415926
            public static final someFinalMethod( ) { …}
     }
```

this in Java

- this refers to the current object.

```
public class Point
{
  public int x = 0;
  public int y = 0;
  //constructor-1
  public Point(int a, int b)
  {
     this.x = a; // means x = a;
     this.y = b; // means y = b;
  }
// constructor-2
public Point ( )
 {
     this (0, 0); // call constructor-1 with (0, 0)
 }
}
```

super in Java

- super is a way to call parent's functions/data

```
class Window
{
   private int length = 0;
   private int width = 0;
    //constructor
    public Window (int l, int w)
    {
    this.length = l;
        this.width = w;
    }
    // function printProperties
    public void printProperties ( )
    {
    System.out.println ("Dimensions:
    " length + " " + width);
    }
}
```

```
public class Browser extends Window
{
    private String browserType;
    public Browser(int l, int w )
    {
        // Call parent constructor
        super (l, w);
        browserType = "Firefox";
    }
    public void printProperties( )
    {
        // Call parent function
        super.printProperties( );
        System.out.println ("browserType: "
    + browserType);
    }
 }
```

Abstract class

Abstract classes

- Consider an object of Account.
- It makes sense to have
  - A specific type (e.g., checking) of account
  - Not just a generic account object.
- A user should be able to create
  - Specific object types.
  - NOT generic objects.
- An abstract class is the generic class.
  - Cannot create objects of this class
- Classes derived from the abstract classes are specific objects.
  - Can create objects of the derived classes.

Abstract classes … contd.

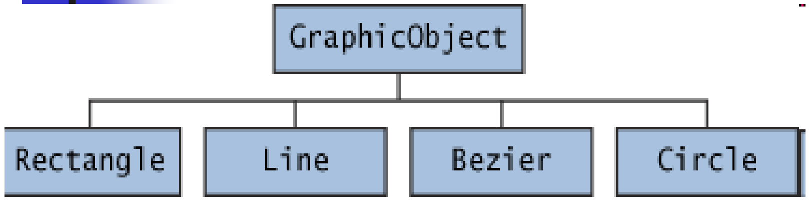
- Abstract class
  - A class that has abstract keyword (prefix)
  - May have the following methods:
    - **abstract** – no implementation, only declaration
    - non-abstract – have implementation
  - Cannot be instantiated
  - Can be extended by (non) abstract subclasses

Abstract class - Example

```
abstract class shape
{
   abstract int findArea( );
   public String showShape( )
     {
         return ("defaultShape");
     }
};
```

```
class square extends shape
{
   private int length;
   public square ( ) { length = -1; }
   public int findArea ( )
     {
        return (length * length);
      }
    public String showShape ( )
     {
        return ("square");
     }
};
```

Abstract class … Example-2



```
abstract class GraphicObject
{
    int x, y;
    // non-abstract method
    // has actual code
    void moveTo (int x1, int y1)
     { … }
    // abstract methods
    // No code or implementation
    abstract void draw(  );
    abstract void resize( );

 }
```

Source: Oracle.com

```
class Circle extends GraphicObject
{
    void draw ( ) { … }
    void resize ( ) { … }
 };

 class Rectangle extends GraphicObject
{
    void draw( ) { … }
    void resize( ) { … }
 } ;
```

Interfaces

Java interfaces

- Interface
  - Similar to abstract class
    - Cannot be instantiated.
  - Difference
    - Member functions can only be defined.
    - No implementation for ANY member function.
  - Derived classes need to implement functions.

Interface … example

```
interface myInterface
{
    void function1( );
     int function2( );
}
```

Note: No implementation for function1 or function2

```
class myClass implements
    myInterface
{
    void function1( )
      {
          System.out.println (“fn1”);
      }
    int function2( )
      {
          System.out.println (“fn2”);
          return (1);
      }
}
```

Multiple inheritance in Java

- Java allows implementation of multiple interfaces.
  - class myClass implements intfc1, intfc2 is allowed
- Java does not allow extension of more than one class.
  - class myClass extends class1, class2 is NOT allowed.
- Extension of one class, implementation of multiple interfaces is allowed.
  - Class myClass extends class1, implements interface1, implements interface2 is allowed.

Packages

Packages

- A way of grouping different (related) classes in Java.
- Java itself provides many packages
  - E.g. Math, I/O, Exception, etc.
- Packages are used to provide
  - Access restrictions
  - Namespace management

How to create packages

- Simply put “package” in the beginning of a class (should be the first line).

```
package example_package
 class myClass1
 {
                 // Code
 }

package example_package
 class myClass2
 {
                 // Code
 }
```

- myClass1 and myClass2 are now part of example_package
- A package typically has many classes.

Creating packages example

```
package graphics;
public interface Draggable { . . . }

package graphics;
public abstract class Graphic { . . . }

package graphics;
public class Circle extends Graphic implements Draggable { . . . }

package graphics;
public class Rectangle extends Graphic implements Draggable { . . . }

package graphics;
public class Point extends Graphic implements Draggable { . . . }
```

Source: oracle.com

Using classes from external packages

- Use **import** keyword.
  - Can import the entire package. E.g.,
    - import java.lang.*;
    - import mypackage.*;
  - Or, can import specific classes in a package
    - import mypackage.myclass;

- E.g. Use math functions.

```
import java.lang.math;
public class myClass
{
   public double computeArea (int r)
    {
             return ( math.PI * r * r);
     }
}
```

Packages … contd.

- Packages can be created, included in a hierarchical way
  - E.g., com. mycompany.mypackage
    - Package from mycompany
  - com.anothercompany.package
    - Package from anothercompany.
  - They can be included as
    - import com.mycompany.mypackage
    - import com.anothercompany.mypackage