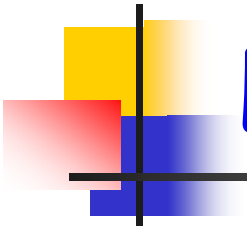


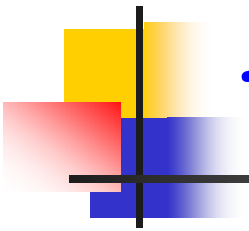
Lecture5

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

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.

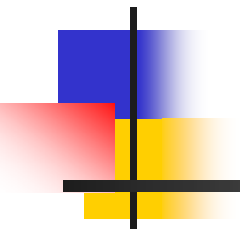
```
public class Point
{
  public int x = 0;
  public int y = 0;
  //constructor-1
  public Point(int a, int b)
  {
     this.x = a; // means x = a;
     this.y = b; // means y = b;
  }
// constructor-2
public Point ( )
  {
     this (0, 0); // call constructor-1 with (0, 0)
  }
}
```

super in Java

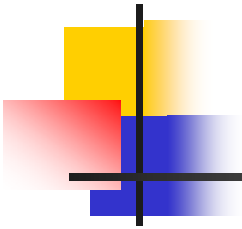
- super is a way to call parent's functions/data

```
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);
    }
 }
```

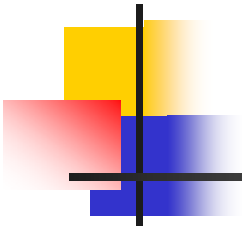


Abstract class



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.

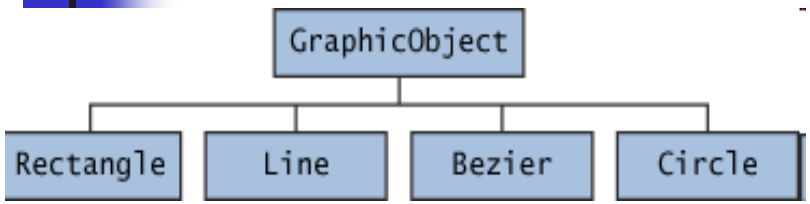
- Abstract class
  - A class that has abstract keyword (prefix)
  - May have the following methods:
    - abstract – no implementation, only declaration
    - non-abstract – have implementation
  - Cannot be instantiated
  - Can be extended by (non) abstract subclasses

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



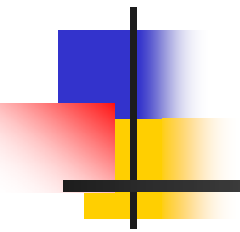

```
abstract class GraphicObject
{
    int x, y;
    // non-abstract method
    // has actual code
    void moveTo (int x1, int y1)
     { ... }
    // abstract methods
    // No code or implementation
    abstract void draw(  );
    abstract void resize( );

 }
```

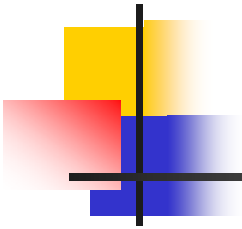
```
class Circle extends GraphicObject
{
    void draw ( ) { ... }
    void resize ( ) { ... }
 };
```

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

Source: Oracle.com

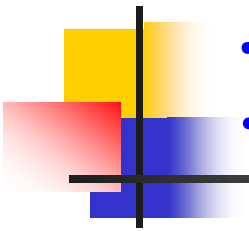


Interfaces



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

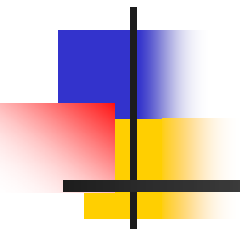
```
interface myInterface
{
    void function1( );
     int function2( );
}
```

Note: No implementation for function1 or function2

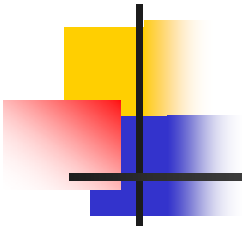
```
class myClass implements
    myInterface
{
    void function1( )
      {
          System.out.println ("fn1");
      }
    int function2( )
      {
          System.out.println ("fn2");
          return (1);
      }
}
```

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.



Exceptions



Exceptions

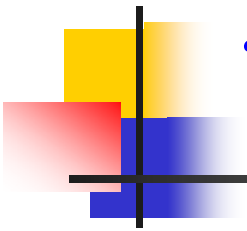
- A way to handle error or unexpected conditions.
- Used to ensure that error conditions are handled gracefully.
- In Java, there is a class called Exception that is used to handle any generic exceptional condition.
  - Exception is derived from Throwable
- Many kinds of specific exceptions are also available
  - I/O exceptions
  - Array out of bound exceptions
  - Class not found exception
  - No such method exception
- Users can define their own exceptions
  - Derive their class from Exception

How to catch exceptions

- Use try catch statements

```
try
  {
    // some code
   }
catch (AnyException e)
  {
    // Error handling code
  }
```

- AnyException is any Exception
  - Can be Java defined exception, or user defined



Throwing Exceptions

- Throw an exception using

  throw (e);

- Any exception – user defined or Java defined exception (e) can be thrown.

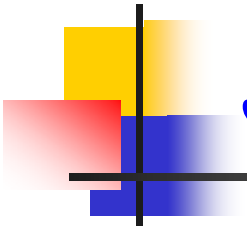
Throwing exceptions … contd.

- The function that throws any exception should define it.

```
void function1( ) throws AnyException
    {
       // code
            // in case of error conditions
       throw (new AnyException( ) );
    }
```

- Any function calling myFunction should catch the exception

```
void function2 ( )
    {
       try
            {
               function1( )
            }
       catch (AnyException e) // Or parent of AnyException
            {
              // Error handling code
            }
     }
```



Java – finally

- finally – a way to handle any left over (cleanup) issues.
- Should be present in the end, after try and catch are done.
- Typically used to clean up resources, open files, file descriptors, sockets, etc.