Interfaces

Forest Agostinelli
University of South Carolina

Interfaces

- Similar to a Class
  - Creates a Type
  - The identifier of an interface MUST match the filename
- Defines the functionality (methods) a class MUST *implement*
- Creates a non-constructible Type
  - Can only construct Classes that *implement* an interface
  - Classes that *implement* an interface can be assigned to variables of that interface type
- Only Contains method signatures
  - No method body or functionality
  - No instance variables
- “Blueprints for Classes”

Creating an Interface Syntax

```
public interface <<id>>
{
	<<method signatures>>;
}
```

Example

```
public interface Shape
{
	public void setHSpace(int aH);
	public int getHSpace();
	public void drawShape();
	public void drawShapeAt(int lineNumber);
}
```

Interfaces

- Reserved word “implements” is used between a class and an interface
- If a method is not defined in a class that *implements* an interface then the class will have a syntax error
- Useful for when the functionality of a class can be done in a variety of ways

Class using an Interface Syntax

```
public class <<class id>> implements <<interface id>>
{
      <<methods from the interface must be defined in this class>>
}
```

Example

```
public class BasicShape implements Shape
{
      //Methods setHSpace, getHSpace, drawShape,
      //and drawShapeAt must be defined in here

}
```

Interfaces

- Declaring a variable of an interface-type is the same as declaring a variable of a class-type
  - Type followed by an identifier
  - Identifiers have the same rules as every other variable identifier
- Cannot construct an instance (object) of an interface
  - Interfaces are non-constructible types
- Only Classes that *implements* the interface can be constructed an assigned

Using an Interface as a Type Syntax

```
//Declaring a variable using the interface as a type
<<interface id>> <<id>>;
//Creating an instance of class that uses the interface
<<id>> = new <<Class Constructor>>;
```

Example

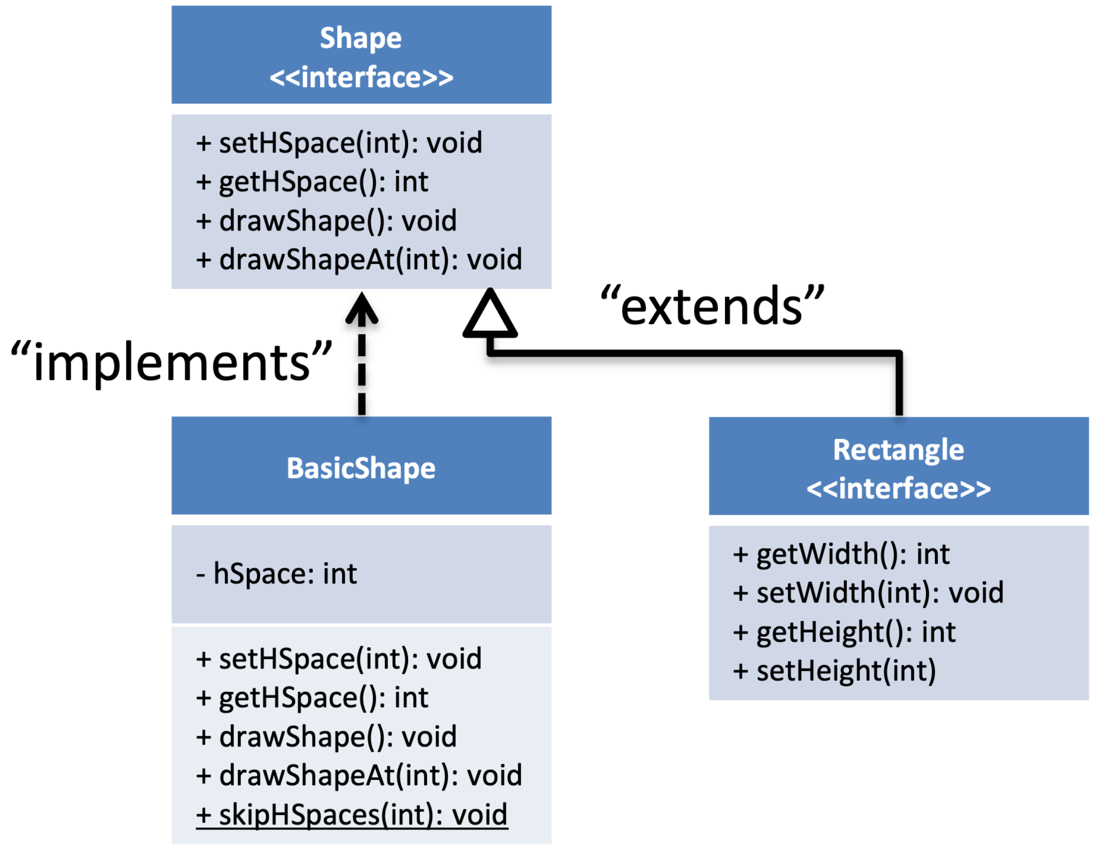
```
//Correct
Shape s = new BasicShape();
//Incorrect, because interfaces cannot be constructed
Shape s2 = new Shape();//Syntax error here
```

Shapes

- Problem: We must create a program that can draw a variety of shapes in the console
- Draw Shapes in the console at set locations
  - Horizontal Spacing
  - Vertical Spacing
- Some Shapes mentioned were:
  - Rectangle
  - Triangle
  - Maybe more?

- Shapes could be drawn in a variety of ways
  - Filled
  - Hollow
  - Upside Down Triangle
  - Checkered Rectangle
  - Horizontal Striped Rectangle
  - Vertical Striped Rectangle
  - Etc.

Shapes

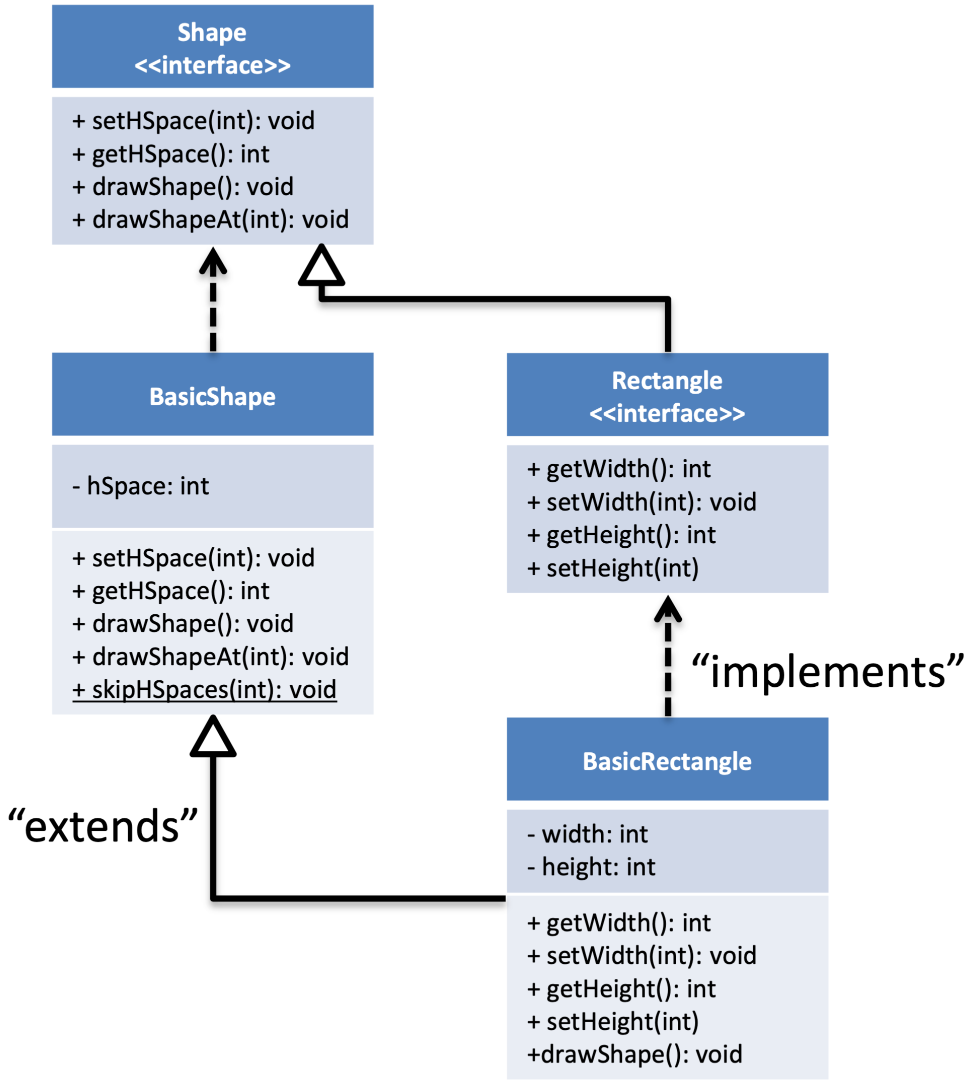

Shape
<<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
"implements"
"extends"
BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Rectangle
<<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)

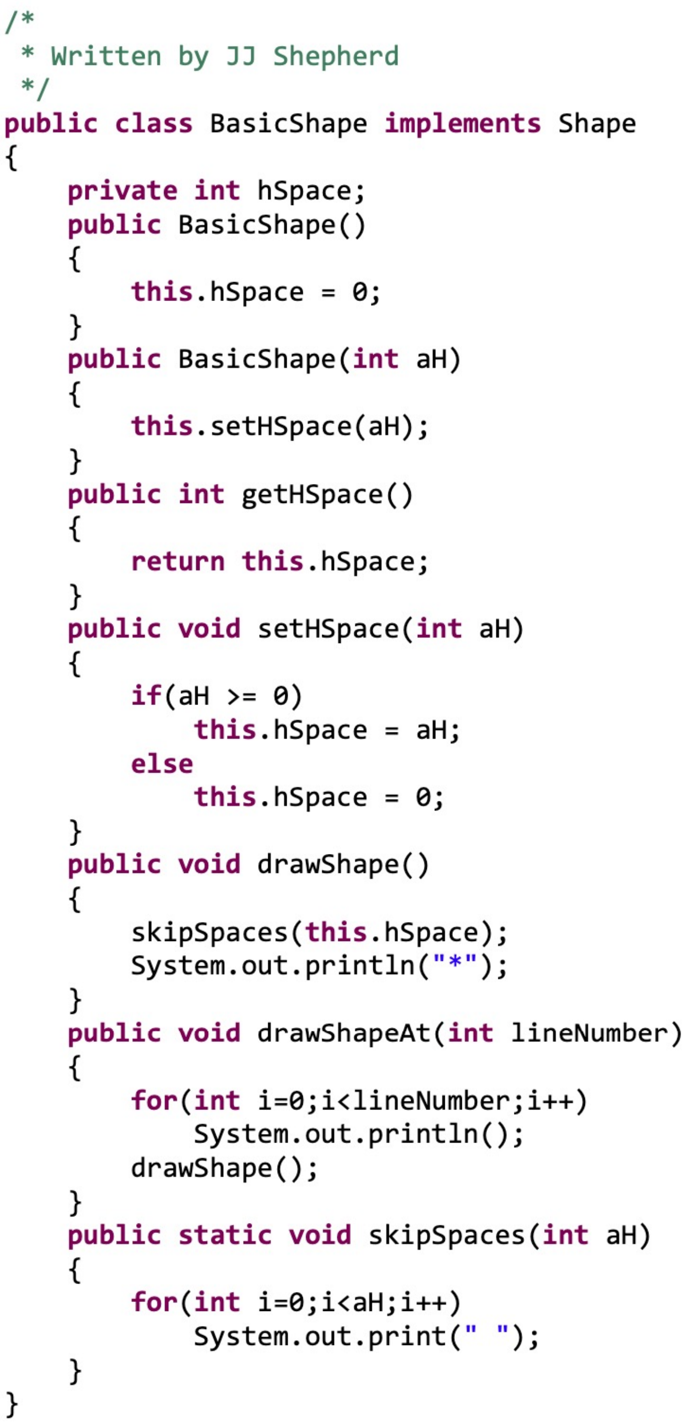
```java
/*
 * Written by JJ Shepherd
 */
public interface Shape {
    public void setHSpace(int aH);
    public int getHSpace();
    public void drawShape();
    public void drawShapeAt(int lineNumber);
}
```

```java
/*
 * Written by JJ Shepherd
 */
public interface Rectangle extends Shape
{
    public int getWidth();
    public int getLength();
    public void setWidth(int aW);
    public void setLength(int aL);
}
```

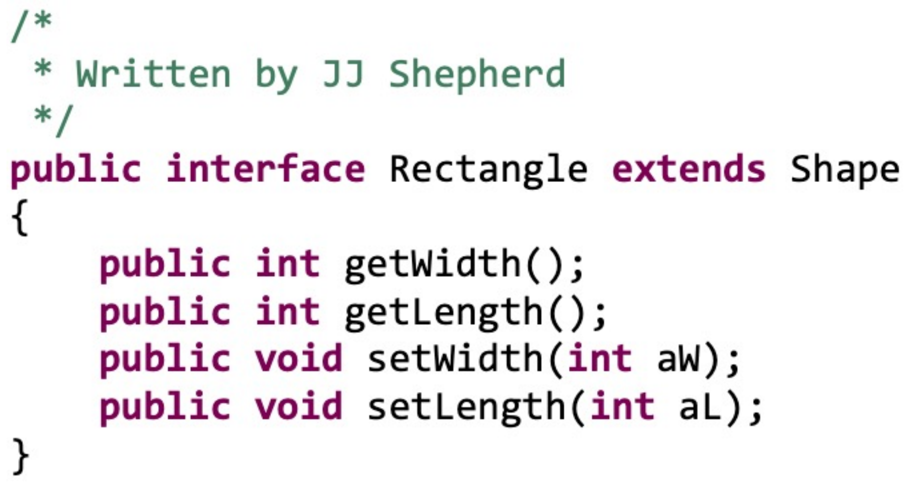
```java
/*
 * Written by JJ Shepherd
 */
public class BasicShape implements Shape
{
    private int hSpace;
    public BasicShape()
    {
        this.hSpace = 0;
    }
    public BasicShape(int aH)
    {
        this.setHSpace(aH);
    }
    public int getHSpace()
    {
        return this.hSpace;
    }
    public void setHSpace(int aH)
    {
        if(aH >= 0)
            this.hSpace = aH;
        else
            this.hSpace = 0;
    }
    public void drawShape()
    {
        skipSpaces(this.hSpace);
        System.out.println("*");
    }
    public void drawShapeAt(int lineNumber)
    {
        for(int i=0;i<lineNumber;i++)
            System.out.println();
        drawShape();
    }
    public static void skipSpaces(int aH)
    {
        for(int i=0;i<aH;i++)
            System.out.print(" ");
    }
}
```

Shapes

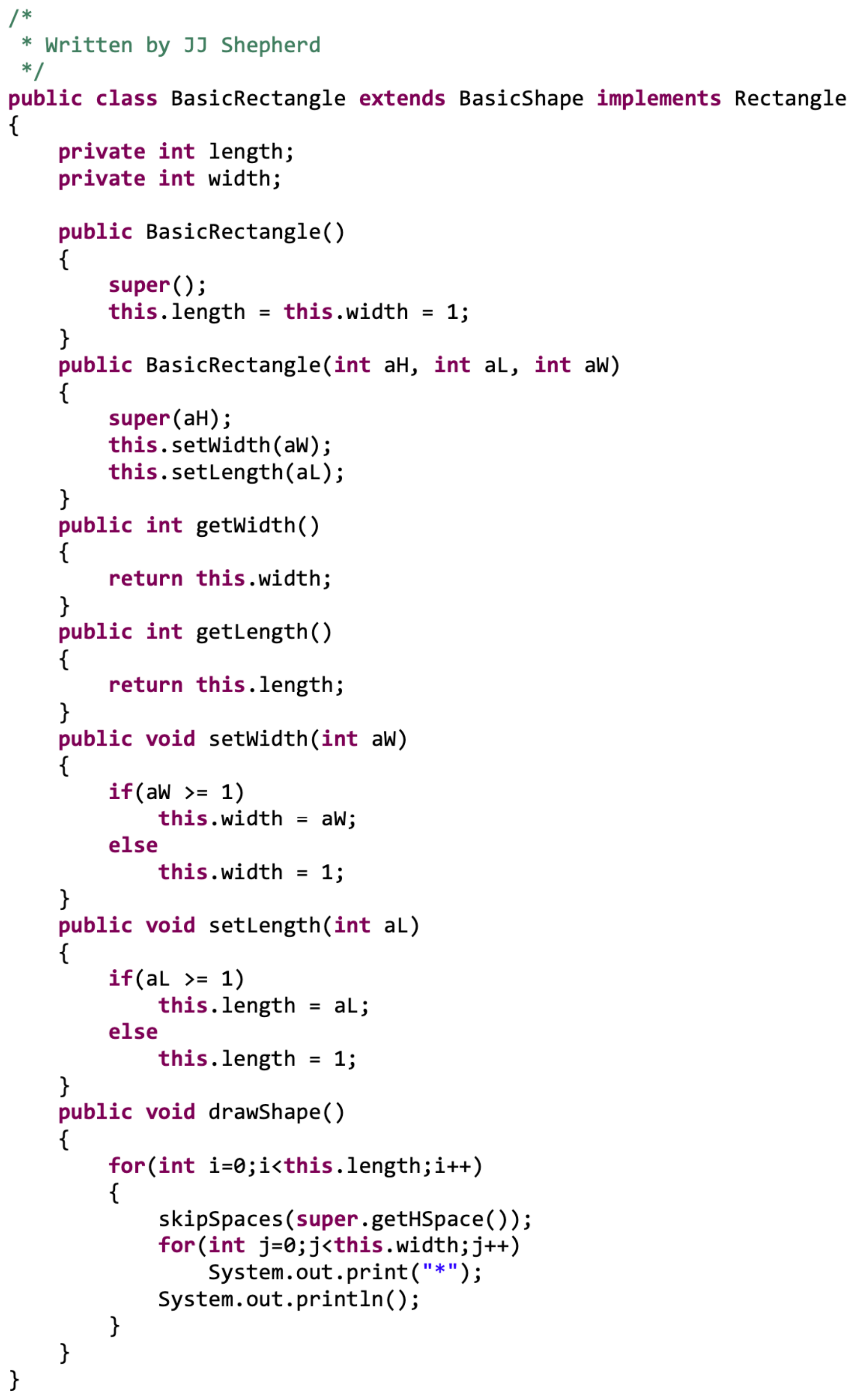

Shape
<<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Rectangle
<<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
"implements"
BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
"extends"



```java
/*
 * Written by JJ Shepherd
 */
public class BasicShape implements Shape
{
    private int hSpace;
    public BasicShape()
    {
        this.hSpace = 0;
    }
    public BasicShape(int aH)
    {
        this.setHSpace(aH);
    }
    public int getHSpace()
    {
        return this.hSpace;
    }
    public void setHSpace(int aH)
    {
        if(aH >= 0)
            this.hSpace = aH;
        else
            this.hSpace = 0;
    }
    public void drawShape()
    {
        skipSpaces(this.hSpace);
        System.out.println("*");
    }
    public void drawShapeAt(int lineNumber)
    {
        for(int i=0;i<lineNumber;i++)
            System.out.println();
        drawShape();
    }
    public static void skipSpaces(int aH)
    {
        for(int i=0;i<aH;i++)
            System.out.print(" ");
    }
}
```

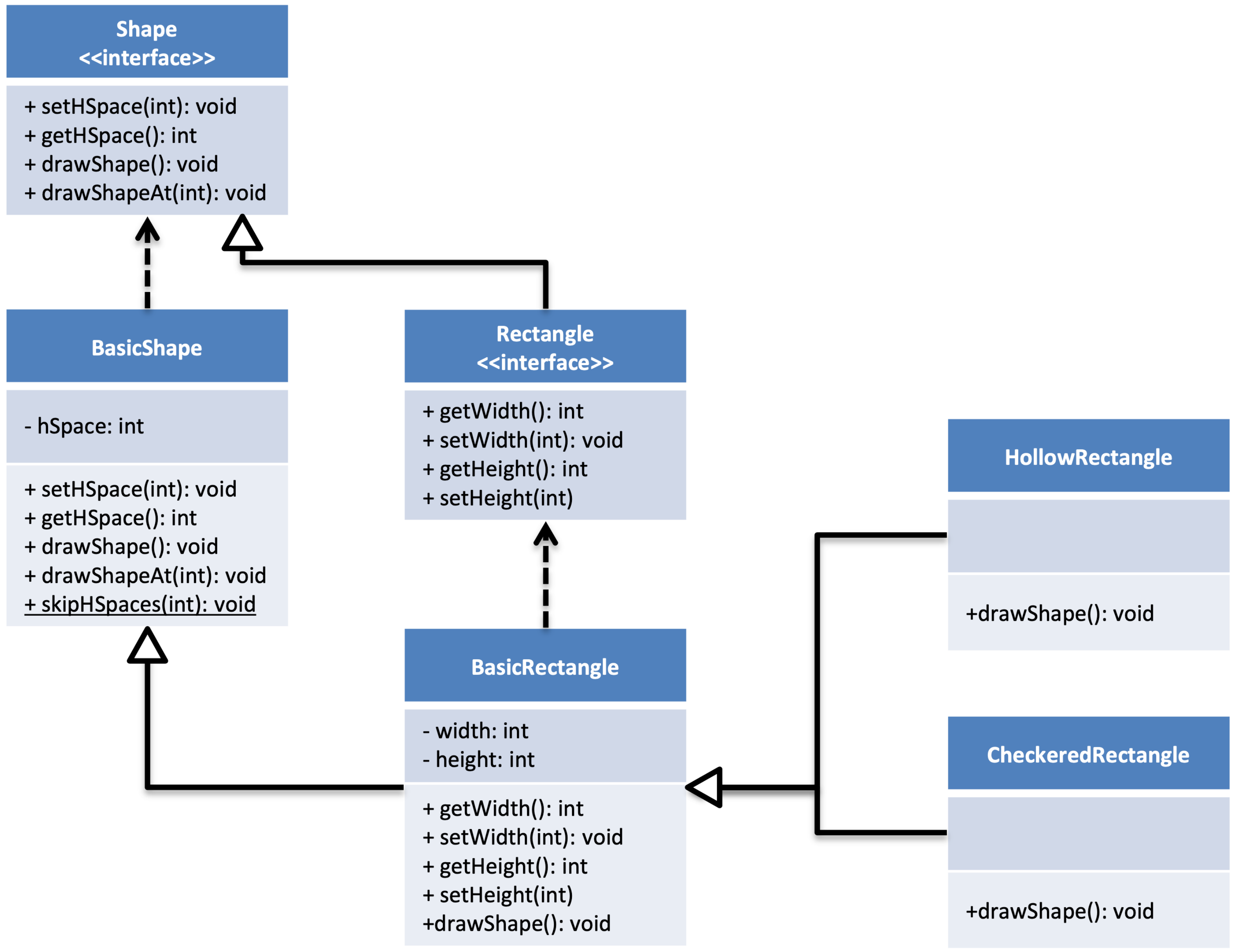


```java
/*
 * Written by JJ Shepherd
 */
public interface Rectangle extends Shape
{
    public int getWidth();
    public int getLength();
    public void setWidth(int aW);
    public void setLength(int aL);
}
```



```java
/*
 * Written by JJ Shepherd
 */
public class BasicRectangle extends BasicShape implements Rectangle
{
    private int length;
    private int width;

    public BasicRectangle()
    {
        super();
        this.length = this.width = 1;
    }
    public BasicRectangle(int aH, int aL, int aW)
    {
        super(aH);
        this.setWidth(aW);
        this.setLength(aL);
    }
    public int getWidth()
    {
        return this.width;
    }
    public int getLength()
    {
        return this.length;
    }
    public void setWidth(int aW)
    {
        if(aW >= 1)
            this.width = aW;
        else
            this.width = 1;
    }
    public void setLength(int aL)
    {
        if(aL >= 1)
            this.length = aL;
        else
            this.length = 1;
    }
    public void drawShape()
    {
        for(int i=0;i<this.length;i++)
        {
            skipSpaces(super.getHSpace());
            for(int j=0;j<this.width;j++)
                System.out.print("*");
            System.out.println();
        }
    }
}
```

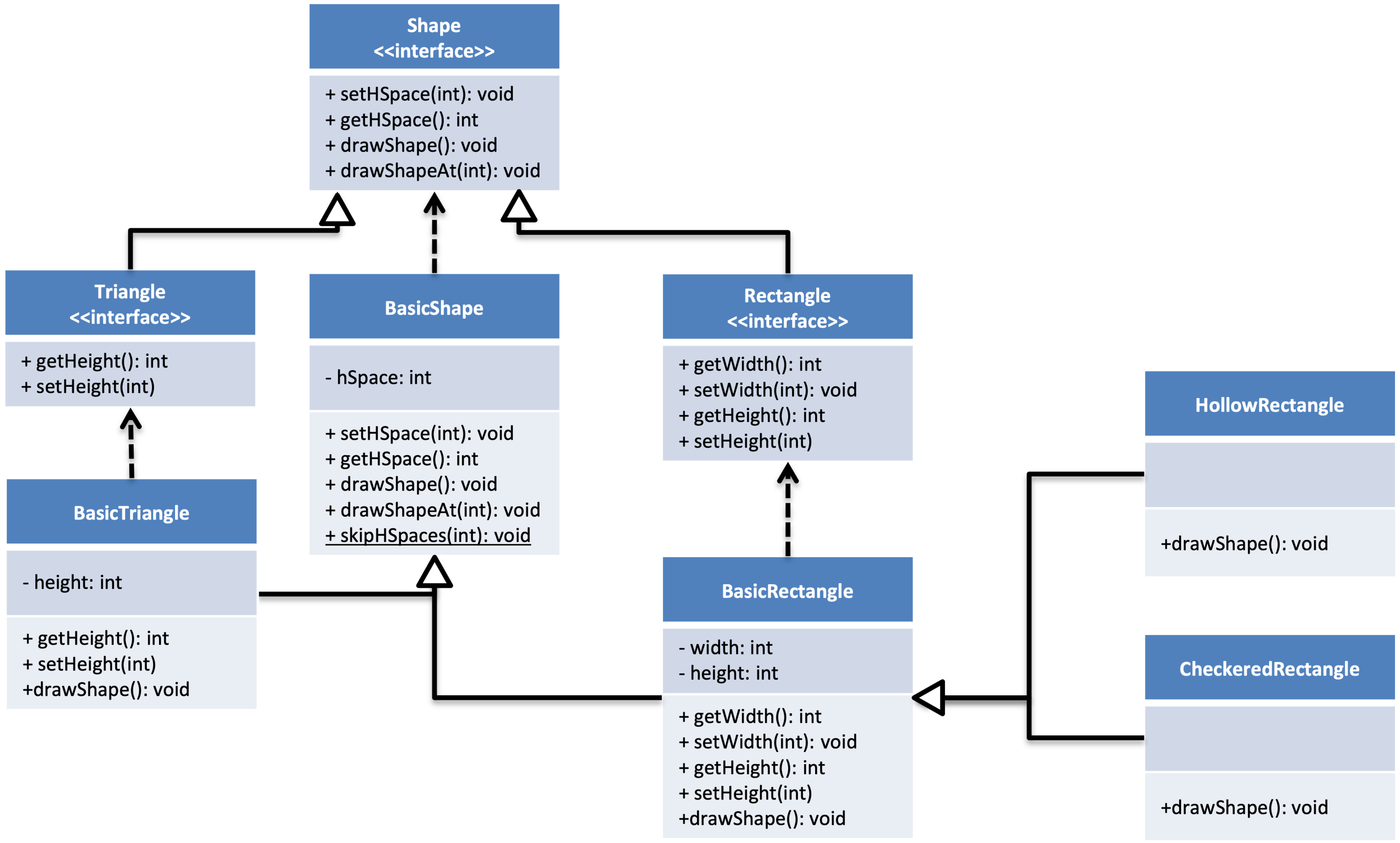
Shapes


Shape
<<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Rectangle
<<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
HollowRectangle
+drawShape(): void
CheckeredRectangle
+drawShape(): void

```java
/*
 * Written by JJ Shepherd
 */
public class BasicRectangle extends BasicShape implements Rectangle
{
    private int length;
    private int width;

    public BasicRectangle()
    {
        super();
        this.length = this.width = 1;
    }
    public BasicRectangle(int aH, int aL, int aW)
    {
        super(aH);
        this.setWidth(aW);
        this.setLength(aL);
    }
    public int getWidth()
    {
        return this.width;
    }
    public int getLength()
    {
        return this.length;
    }
    public void setWidth(int aW)
    {
        if(aW >= 1)
            this.width = aW;
        else
            this.width = 1;
    }
    public void setLength(int aL)
    {
        if(aL >= 1)
            this.length = aL;
        else
            this.length = 1;
    }
    public void drawShape()
    {
        for(int i=0;i<this.length;i++)
        {
            skipSpaces(super.getHSpace());
            for(int j=0;j<this.width;j++)
                System.out.print("*");
            System.out.println();
        }
    }
}
```

```java
/*
 * Written by JJ Shepherd
 */
public class HollowRectangle extends BasicRectangle
{

    public HollowRectangle()
    {
        super();
    }
    public HollowRectangle(int aH, int aL, int aW)
    {
        super(aH,aL,aW);
    }
    public void drawShape()
    {
        drawLine();
        drawSides();
        drawLine();
    }
    public void drawLine()
    {
        skipSpaces(super.getHSpace());
        for(int i=0;i<super.getWidth();i++)
            System.out.print("*");
        System.out.println();
    }
    public void drawSides()
    {
        for(int i=0;i<super.getLength()-2;i++)
        {
            skipSpaces(super.getHSpace());
            System.out.print("*");
            skipSpaces(super.getWidth()-2);
            System.out.print("*");
            System.out.println();
        }
    }
}
```

Shapes

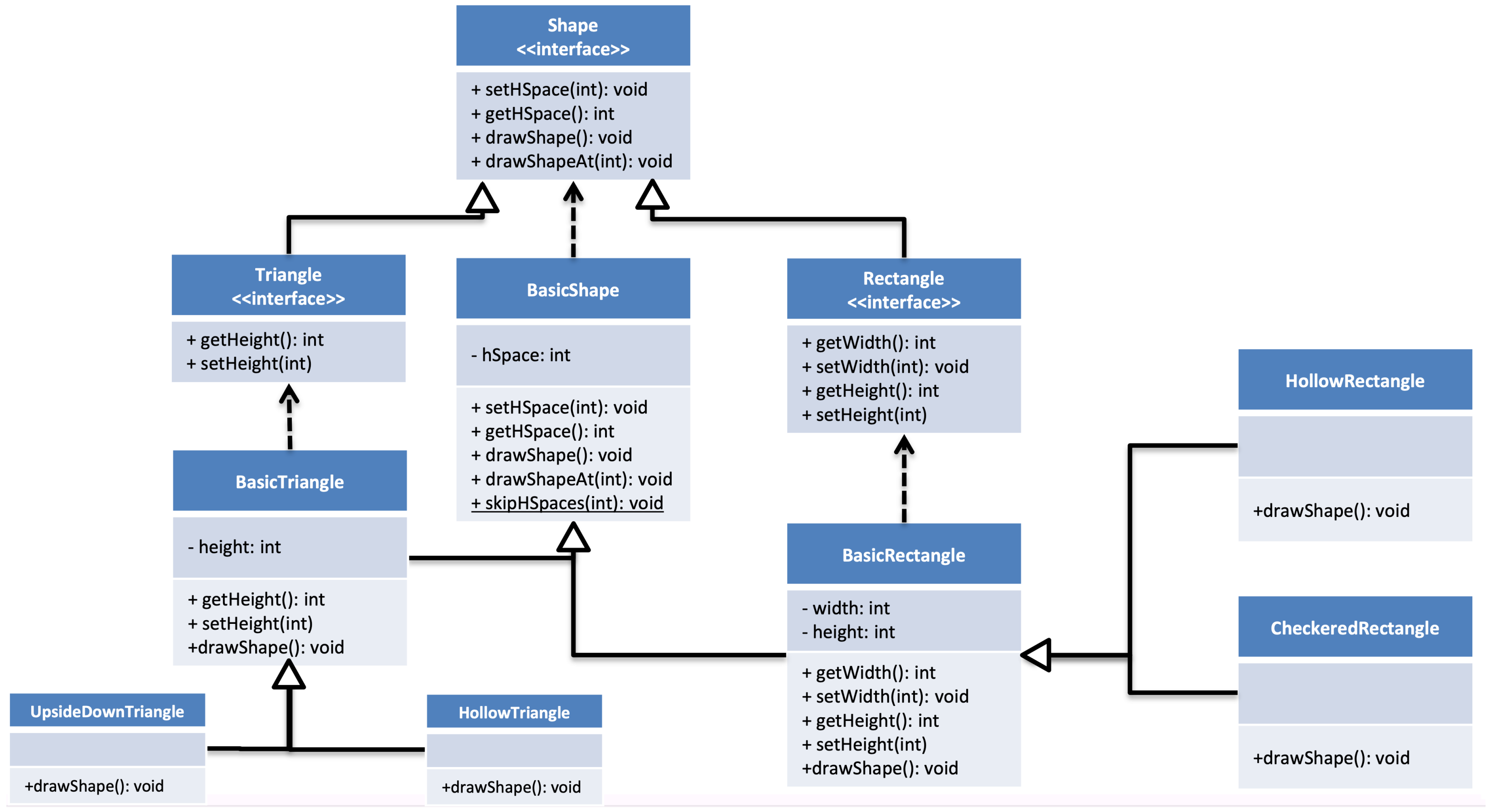

Shape
<<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
Triangle
<<interface>>
+ getHeight(): int
+ setHeight(int)
BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Rectangle
<<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
BasicTriangle
- height: int
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
HollowRectangle
+drawShape(): void
CheckeredRectangle
+drawShape(): void

```java
/*
 * Written by JJ Shepherd
 */
public interface Shape {
    public void setHSpace(int aH);
    public int getHSpace();
    public void drawShape();
    public void drawShapeAt(int lineNumber);
}
```

```java
/*
 * Written by JJ Shepherd
 */
public interface Triangle extends Shape
{
    public int getHeight();
    public void setHeight(int aHe);
}
```

```java
/*
 * Written by JJ Shepherd
 */
public class BasicTriangle extends BasicShape implements Triangle{
    private int height;
    public BasicTriangle()
    {
        super();
        this.height = 1;
    }
    public BasicTriangle(int aH, int aHe)
    {
        super(aH);
        this.setHeight(aHe);
    }
    public int getHeight()
    {
        return height;
    }
    public void setHeight(int aHe)
    {
        if(aHe >= 1)
            this.height = aHe;
        else
            this.height = 1;
    }
    public void drawShape()
    {
        for(int i=0;i<this.height;i++)
        {
            skipSpaces(super.getHSpace());
            for(int j=0;j<i+1;j++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Shapes

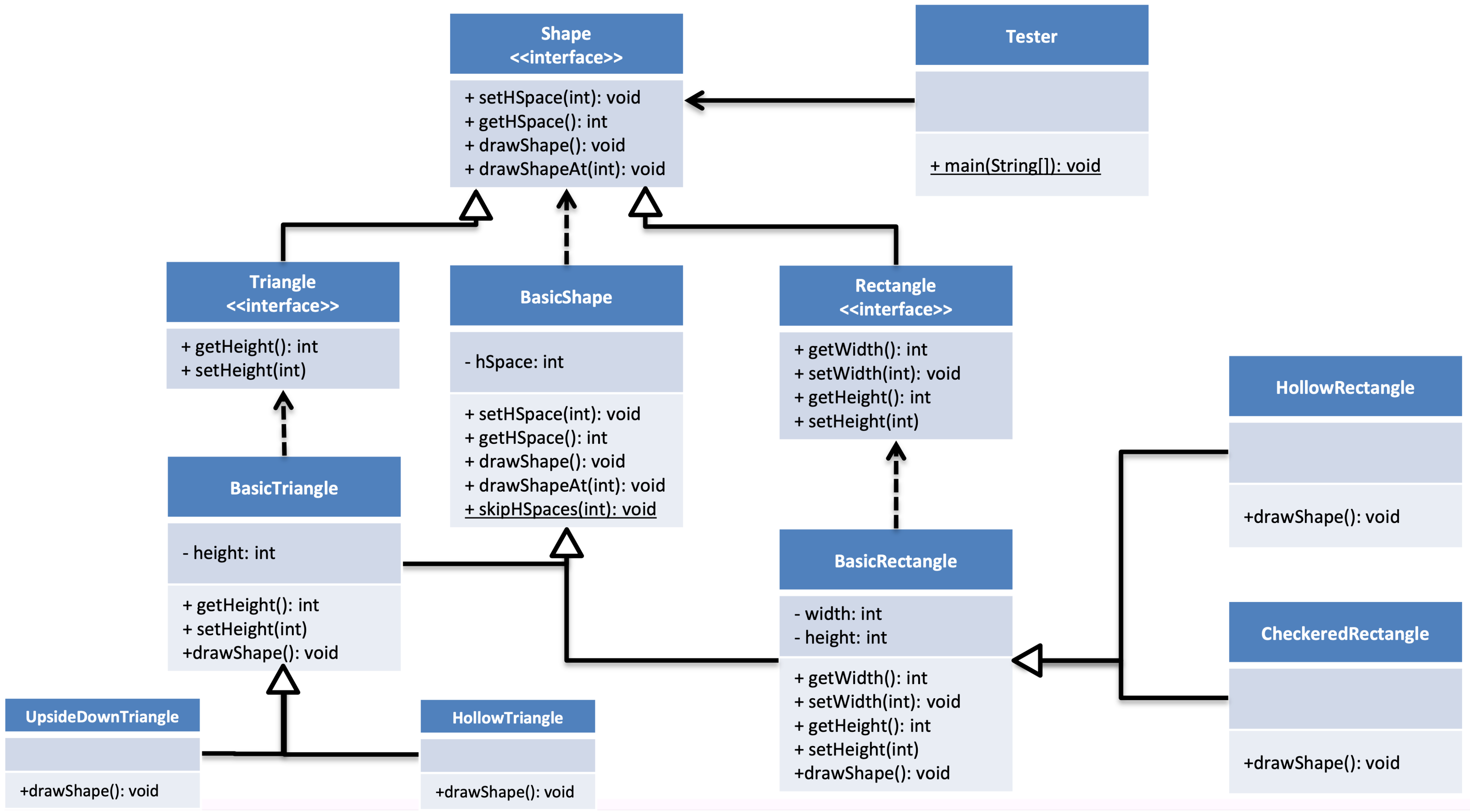


Shape
<<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
Triangle
<<interface>>
+ getHeight(): int
+ setHeight(int)
BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Rectangle
<<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
HollowRectangle
+drawShape(): void
BasicTriangle
- height: int
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
CheckeredRectangle
+drawShape(): void
UpsideDownTriangle
+drawShape(): void
HollowTriangle
+drawShape(): void

```java
/*
 * Written by JJ Shepherd
 */
public class HollowTriangle extends BasicTriangle{
    public HollowTriangle()
    {
        super();
    }
    public HollowTriangle(int aH, int aHe)
    {
        super(aH,aHe);
    }
    public void drawShape()
    {
        //Top point
        skipSpaces(super.getHSpace());
        System.out.println("*");
        int innerSpaces = 0;
        for(int i=0;i<super.getHeight()-2;i++)
        {
            skipSpaces(super.getHSpace());
            System.out.print("*");
            skipSpaces(innerSpaces);
            innerSpaces++;
            System.out.println("*");
        }
        //Bottom Line
        skipSpaces(super.getHSpace());
        for(int i=0;i<super.getHeight();i++)
        {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

```java
/*
 * Written by JJ Shepherd
 */
public class UpsideDownTriangle extends BasicTriangle{
    public UpsideDownTriangle()
    {
        super();
    }
    public UpsideDownTriangle(int aH, int aHe)
    {
        super(aH,aHe);
    }
    public void drawShape()
    {
        for(int i=0;i<super.getHeight();i++)
        {
            skipSpaces(super.getHSpace());
            for(int j=i;j<super.getHeight();j++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Shapes

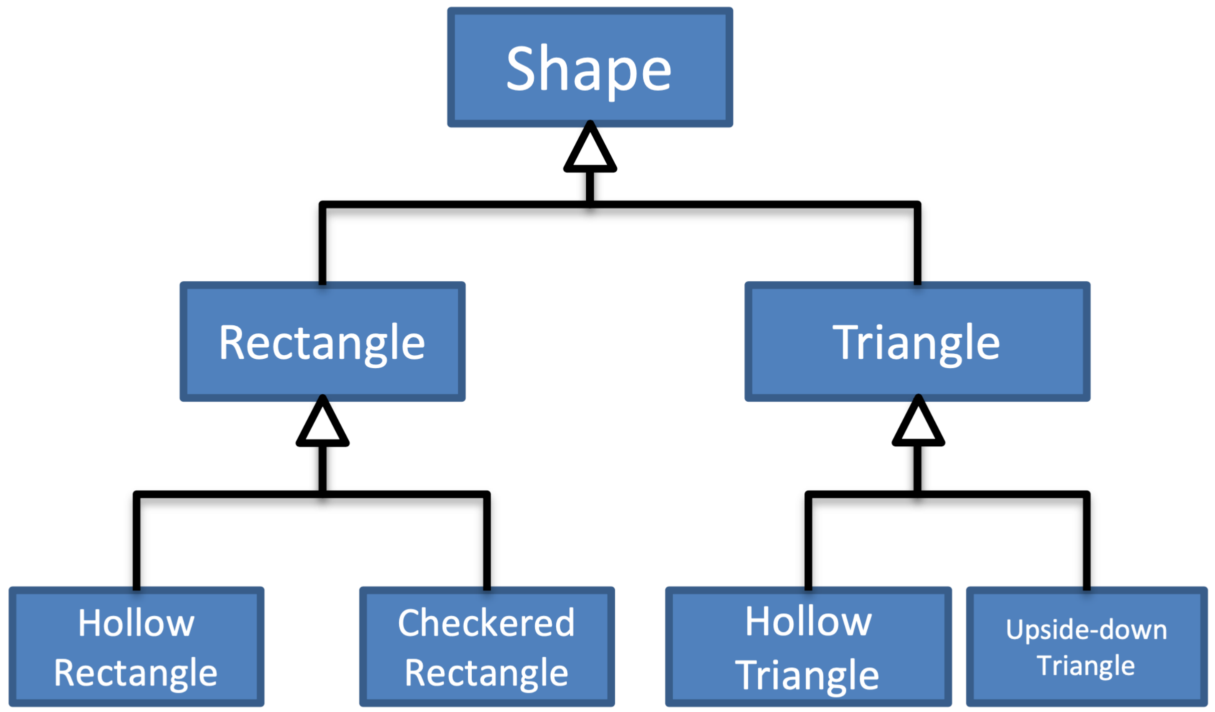

Shape
<<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
Tester
+ main(String[]): void
Triangle
<<interface>>
+ getHeight(): int
+ setHeight(int)
BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Rectangle
<<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
HollowRectangle
+drawShape(): void
BasicTriangle
- height: int
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+drawShape(): void
CheckeredRectangle
+drawShape(): void
UpsideDownTriangle
+drawShape(): void
HollowTriangle
+drawShape(): void

```java
/*
 * Written by JJ Shepherd
 */
public class Tester {

    public static void main(String[] args) {
        //Shape s = new Shape();
        Shape s = new BasicShape();
        s.drawShape();

        Shape[] shapes = new Shape[11];
        shapes[0] = new BasicShape();
        shapes[1] = new BasicShape(4);
        shapes[2] = new BasicRectangle(0,2,3);
        shapes[3] = new BasicRectangle(2,3,4);
        shapes[4] = new HollowRectangle(0,4,4);
        shapes[5] = new HollowRectangle(5,5,5);
        shapes[6] = new CheckeredRectangle(0,7,7);
        shapes[7] = new CheckeredRectangle(5,10,10);
        shapes[8] = new BasicTriangle(0,3);
        shapes[9] = new UpsideDownTriangle(3,5);
        shapes[10] = new HollowTriangle(6,7);
        for(int i=0;i<shapes.length;i++)
        {
            if(shapes[i] != null)
            {
                shapes[i].drawShape();
                //shapes[i].drawShapeAt(i);
            }
        }
    }

}
```

Polymorphism

- Keep in mind
  - Classes *extends* Classes
  - Interfaces *extends* Interfaces
  - Classes *implements* Interfaces
- In Java, classes can implement several interfaces but only extend one other class
  - Extends first followed by Implements
  - Each interface that is implemented is separated by a comma
- Polymorphism allows software to be very *extensible*

<u>Polymorphism Concept</u>


Shape
Rectangle
Triangle
Hollow Rectangle
Checkered Rectangle
Hollow Triangle
Upside-down Triangle