Classes and Objects

`Rectangle`

Introduction to OO Programming

The `Rectangle` Class

The `Rectangle` class is a pre-defined class supplied with the Java API. It is part of the **package** `java.awt`.

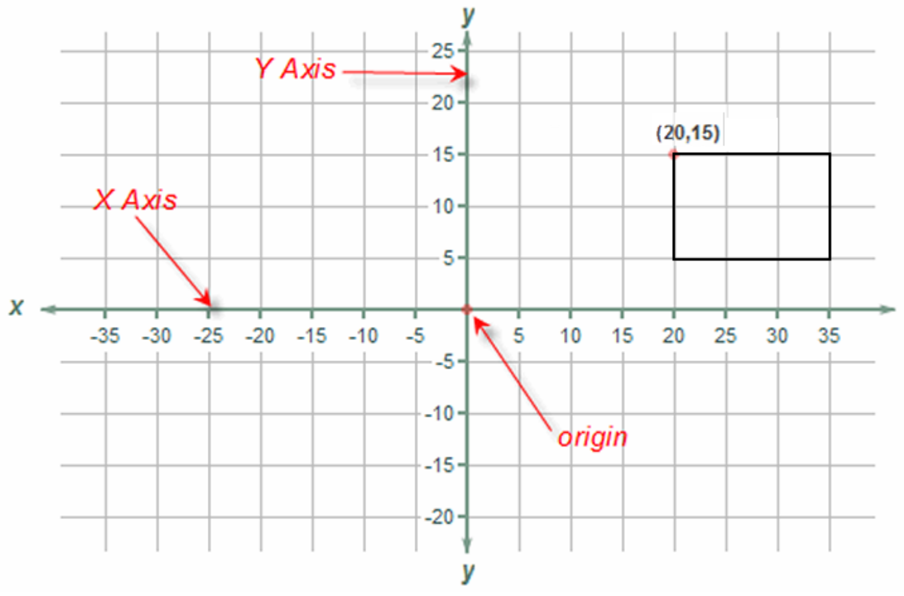
A **package** is a set of classes with a common purpose.

To use the `Rectangle` class in a Java program, you must import it from the `java.awt` package, using the `import` statement at the start of your program.

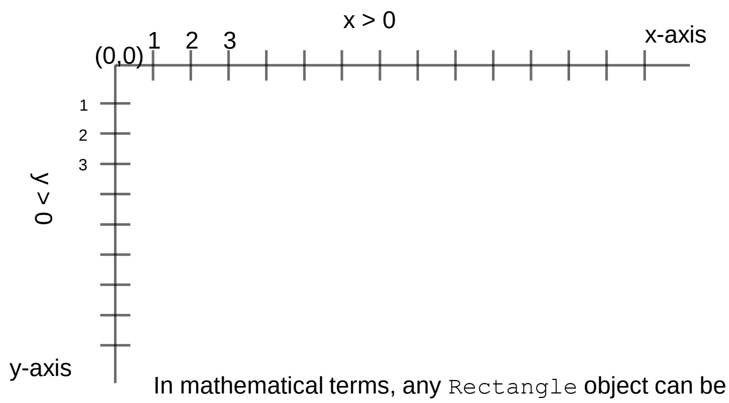
package class

```
import java.awt.Rectangle;
```

The `Rectangle` class models a mathematical representation of a rectangle.



Cartesian Coordinate Plane



In mathematical terms, any `Rectangle` object can be represented by 4 values:

- `x` coordinate, and `y` coordinate,
- `width`,
- `height`.

These are the **properties** of the `Rectangle` class.

The point (0,0) is the top left hand corner of the computer screen.

UML

Unified Modelling Language is a methodology used in designing and documenting Object Oriented programs.

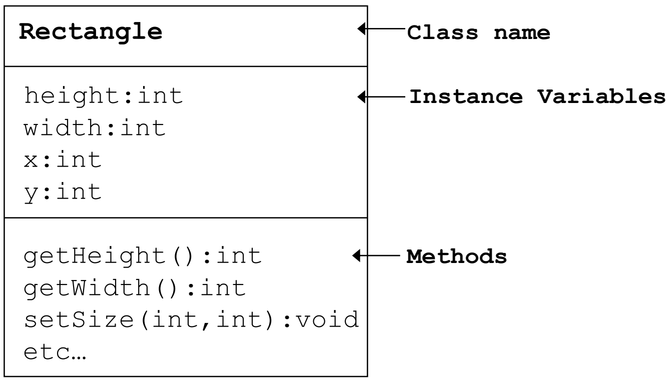
UML includes various diagrams for modelling OO applications.

A UML **class diagram** depicts a class using a box with three sections:

- The **class name** is placed in the top,
- **Instance variables** are placed in the middle,
- **Methods (**and **Constructors)** are placed in the bottom.

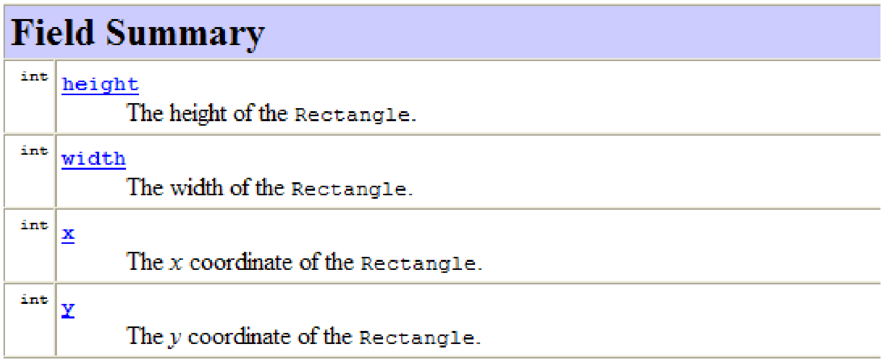
UML Class Diagram for `Rectangle`

The UML class diagram used to represent class `Rectangle` is shown below:



`Rectangle` Instance Variables

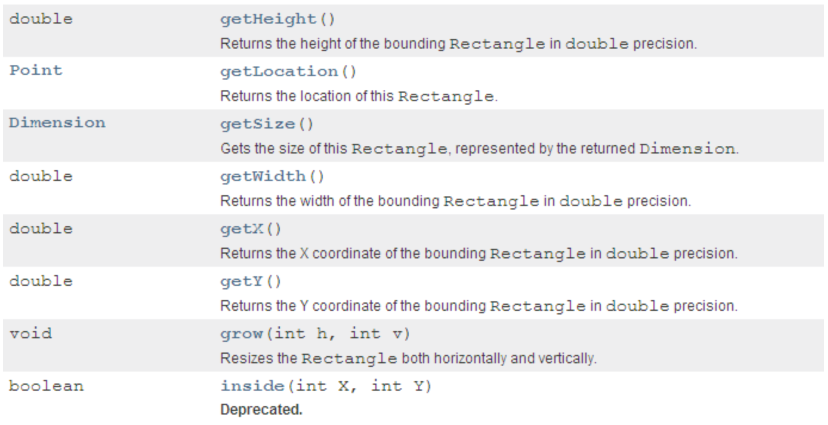
The properties of the `Rectangle` class are represented as **instance variables**. They can be viewed in the Java API by clicking on the field link.



Field Summary	
int	height The height of the `Rectangle`.
int	width The width of the `Rectangle`.
int	x The *x* coordinate of the `Rectangle`.
int	y The *y* coordinate of the `Rectangle`.

`Rectangle` Methods

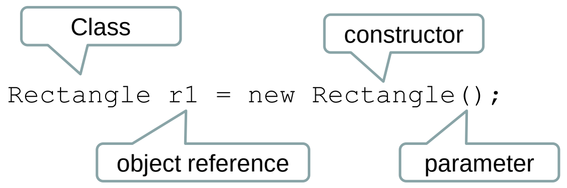
The behaviours of the `Rectangle` class are represented as **methods**. They can be viewed in the Java API by clicking on the method link.



double	`getHeight()` Returns the height of the bounding `Rectangle` in `double` precision.
Point	`getLocation()` Returns the location of this `Rectangle`.
Dimension	`getSize()` Gets the size of this `Rectangle`, represented by the returned `Dimension`.
double	`getWidth()` Returns the width of the bounding `Rectangle` in `double` precision.
double	`getX()` Returns the X coordinate of the bounding `Rectangle` in `double` precision.
double	`getY()` Returns the Y coordinate of the bounding `Rectangle` in `double` precision.
void	`grow(int h, int v)` Resizes the `Rectangle` both horizontally and vertically.
boolean	`inside(int X, int Y)` **Deprecated.**

`Rectangle` Objects

```
// Declare an object called r1 which
// is an instance of the Rectangle class
```



```
Rectangle r1 = new Rectangle();
```

```
// Declare objects called r2 and r3 which
// are both instances of the Rectangle class
Rectangle r2 = new Rectangle();
Rectangle r3 = new Rectangle();
```

Calling Methods on Objects

Once an *object* is created, the programmer calls *methods* on that object to access the *instance variables* or perform operations on the instance variables of that object.

Syntax:

```
objectName.method();
```

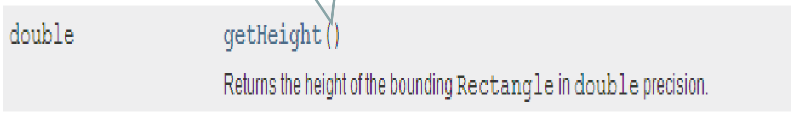
For example:

```
// Call setSize() method on r1
r1.setSize(20,20);
```

Method Parameters

Some methods perform their operations without any inputs. For example, the `getHeight()` method of the `Rectangle` class does not take any input. It can simply be called as is. We can see this in the Java API.

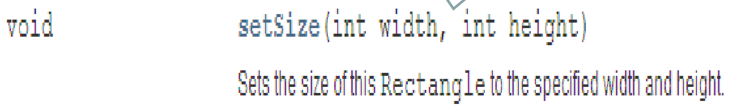
No input parameter required



```
double          getHeight()
                Returns the height of the bounding Rectangle in double precision.
```

Other methods require information from the programmer in order to work. For example, the `setSize()` method of the `Rectangle` class requires two `int` parameters to work. Again, we can see this in the Java API.

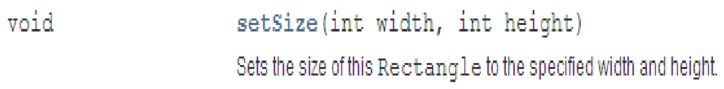
Two input parameters required. Both of type `int`.



```
void            setSize(int width, int height)
                Sets the size of this Rectangle to the specified width and height.
```

The programmer must use the right `type` when passing parameters to a method. Failure to do this will result in either a syntax error, or unexpected results.

The Java API specifies that the `Rectangle setSize()` method expects two `int`s. So it must be called with two `int`s.



```
void                setSize(int width, int height)
                    Sets the size of this Rectangle to the specified width and height.
```

```
r1.setSize(20,20);    //two ints - works fine
r2.setSize(20,23.0); //int & double - compiler error
r2.setSize(20); // ???????  What will happen?
```

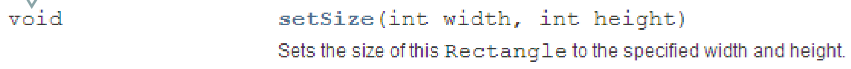
Methods can return values

A **return value** is a value which is returned as the result of a *call* to a method.

Some methods perform operations and do not return any information.

- For example, the `setSize()` method of the `Rectangle` class performs an operation and does not return anything.
- Java indicates that a method does not return a value by having a return type of `void`. We can see this in the Java API.

Return type is `void`. Method does not return a value

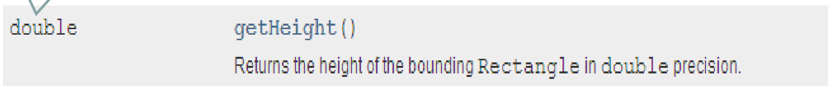


```
void                setSize(int width, int height)
                    Sets the size of this Rectangle to the specified width and height.
```

Other methods return information when invoked (or called).

For example, the `getHeight()` method of the `Rectangle` class returns a value representing the `height` of the object on which it is called. Again, we can see this in the Java API.

Return type is `double`. Method returns a `double` value



```
double          getHeight()
                Returns the height of the bounding Rectangle in double precision.
```

When a method returns a value, we can assign that value to a variable of the appropriate type.

The `getHeight()` method of class `Rectangle` returns a `double` representing the `height` of the `Rectangle` object on which it is called.

We can assign the value returned to a `double` variable.

Assign value returned, to `h`.

Evaluates to `r1`'s `height`.

```
double h = r1.getHeight();
System.out.print("Height is " + h);
```

We can also display the value a method returns using a `System.out.print()` statement.

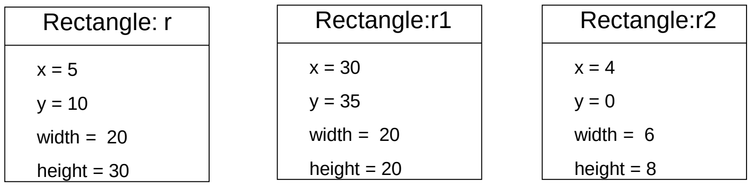
The returned value is passed as an argument to `System.out.print`

Returns `r1`'s `height`.

```
System.out.print("Height is " + r1.getHeight());
```

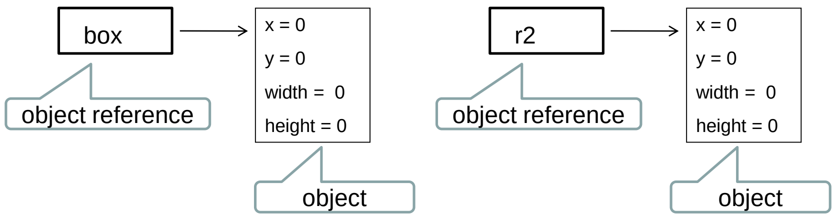
Rectangle Objects

- A `Rectangle` object is not a rectangular shape – it is an object that contains a set of numbers that describe the rectangle



Object vs Object Reference

```
Rectangle box = new Rectangle();
Rectangle r2= new Rectangle();
```



Accessor and Mutator Methods

- `Accessor` method: does not change the state of its implicit parameter (the object it is called on)

```
double w;
w = box.getWidth();
```

- `Mutator` method: changes the state of its implicit parameter

```
box.setSize(1, 5);
```

- Implicit parameter – the object the method is invoked on (i.e. box)