Classes

Introduction

- Objects are particular and finite elements in a larger model.
- The class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).
- A class type variable contains a “handle” (reference) for a specific object.

Class Definition

```
[modifiers] class  name_of_the_class
{
    [variables declaration]
    [constructors declaration]
    [methods declaration]
}
```

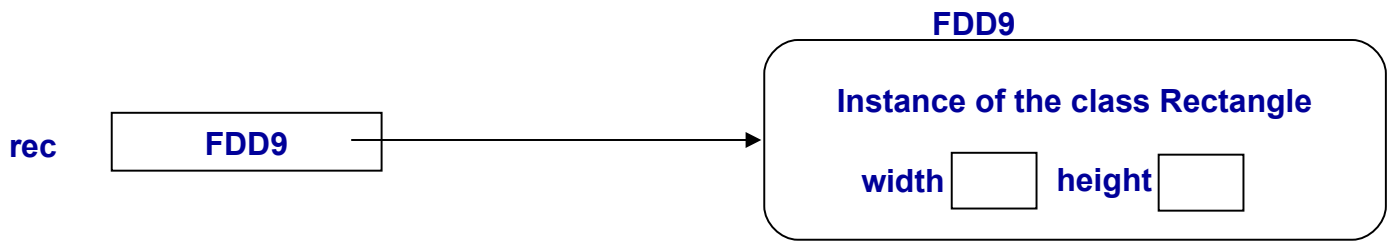
Class Type Variables

- The class type variable isn't an object. It is a simple variable that can hold a reference for a specific object.
- When the class type variable holds a reference of a specific object we can use that variable in order to access the variables inside that object as well as for calling various methods on it.

Class Type Variable

- Given the class `Rectangle`, and a class type variable named `rec`, initializing the `rec` variable might look the following:

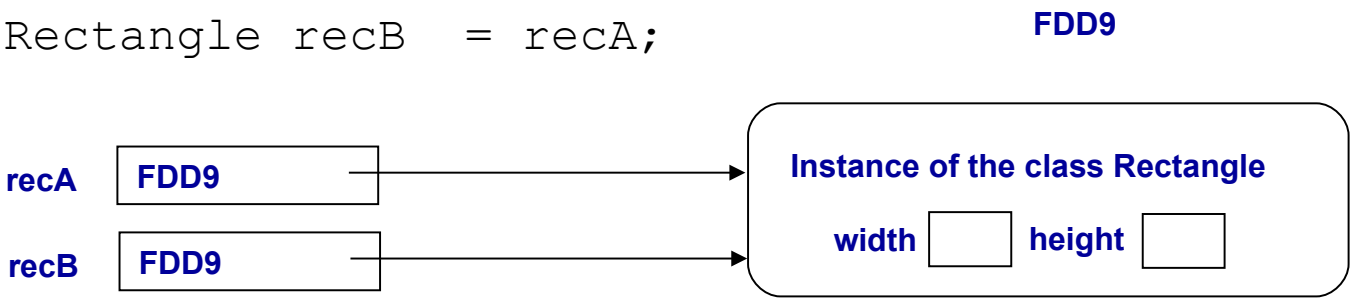
```
Rectangle rec = new Rectangle();
```



Class Type Variable

Given the class `Rectangle`, and the two class type variables named `recA` and `recB`, the following code results in two variables that point to the same object.

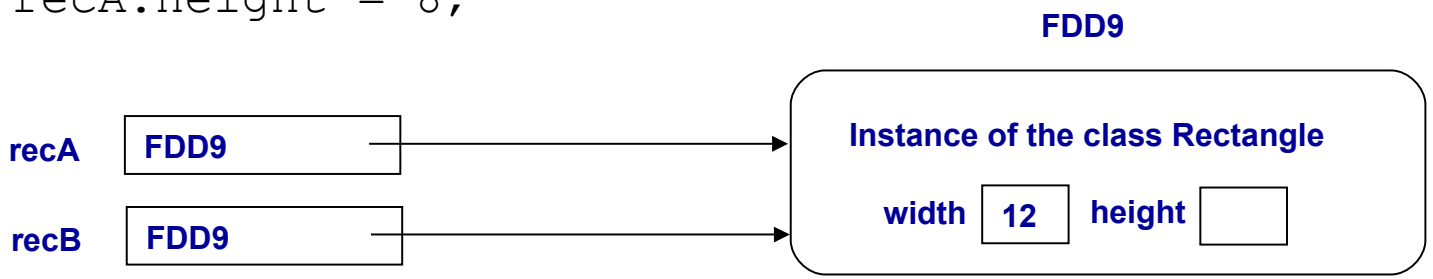
```
Rectangle recA = new Rectangle();
Rectangle recB  = recA;
```



Instance Variables

- We can easily use the dot (.) in order to access the object variables, get their values and set new ones.

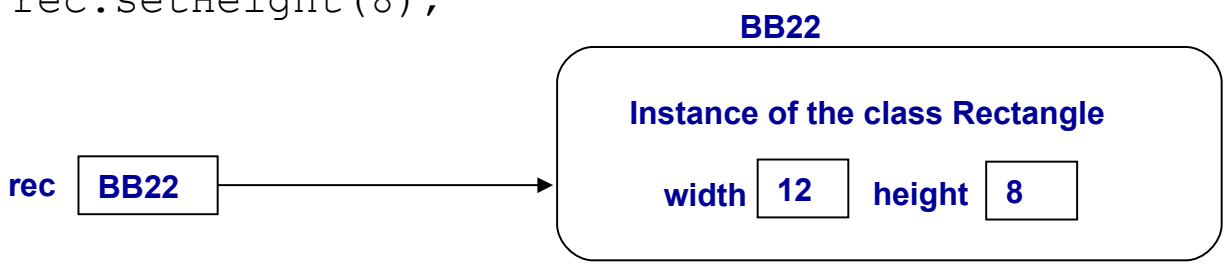
```
Rectangle recA = new Rectangle();
recA.width = 12;
recA.height = 8;
```



Instance Methods

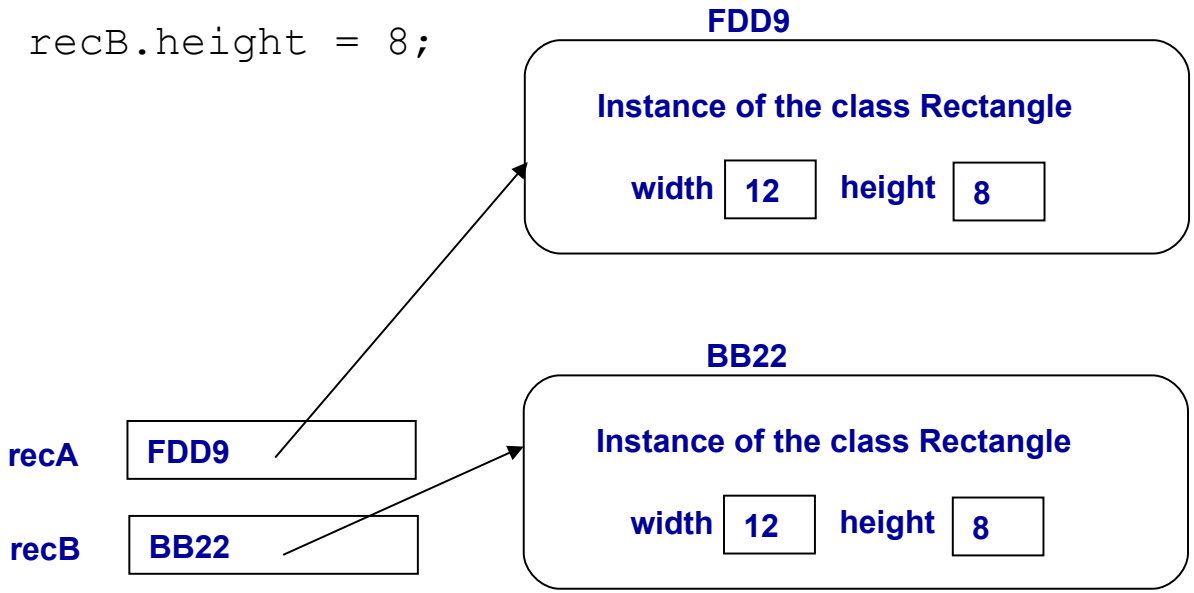
- We can call a method on a specific object by writing the reference for that object following with a dot (.) and the name of the method right after it.

```
Rectangle rec = new Rectangle();
rec.setWidth(12);
rec.setHeight(8);
```



Comparing Objects

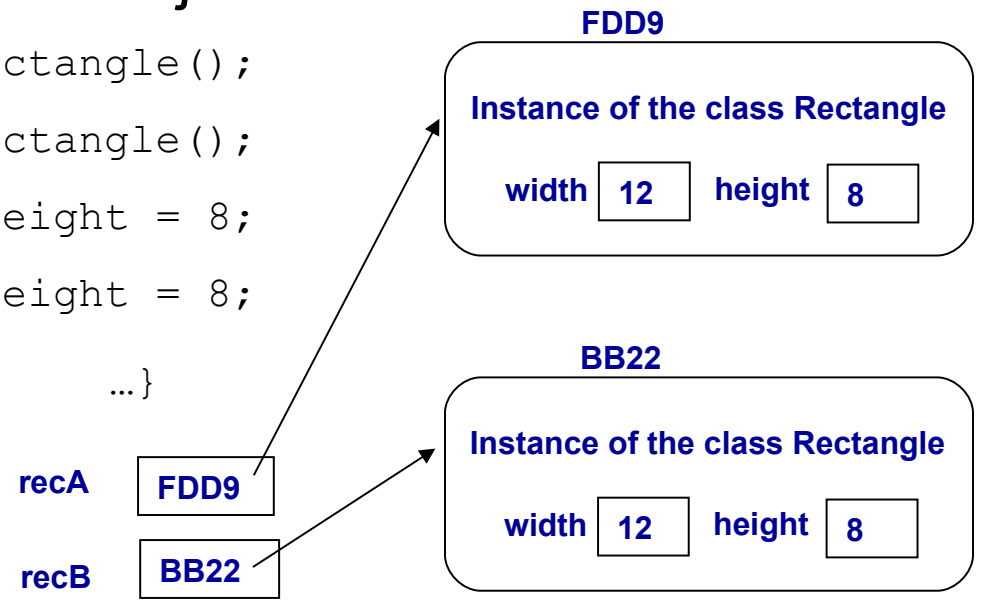
```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA.width = 12; recB.width = 12;
recA.height = 8; recB.height = 8;
if(recA==recB)
{
     ...
}
```



Comparing Objects

- Calling the `equals` method is the right way for comparing between two objects

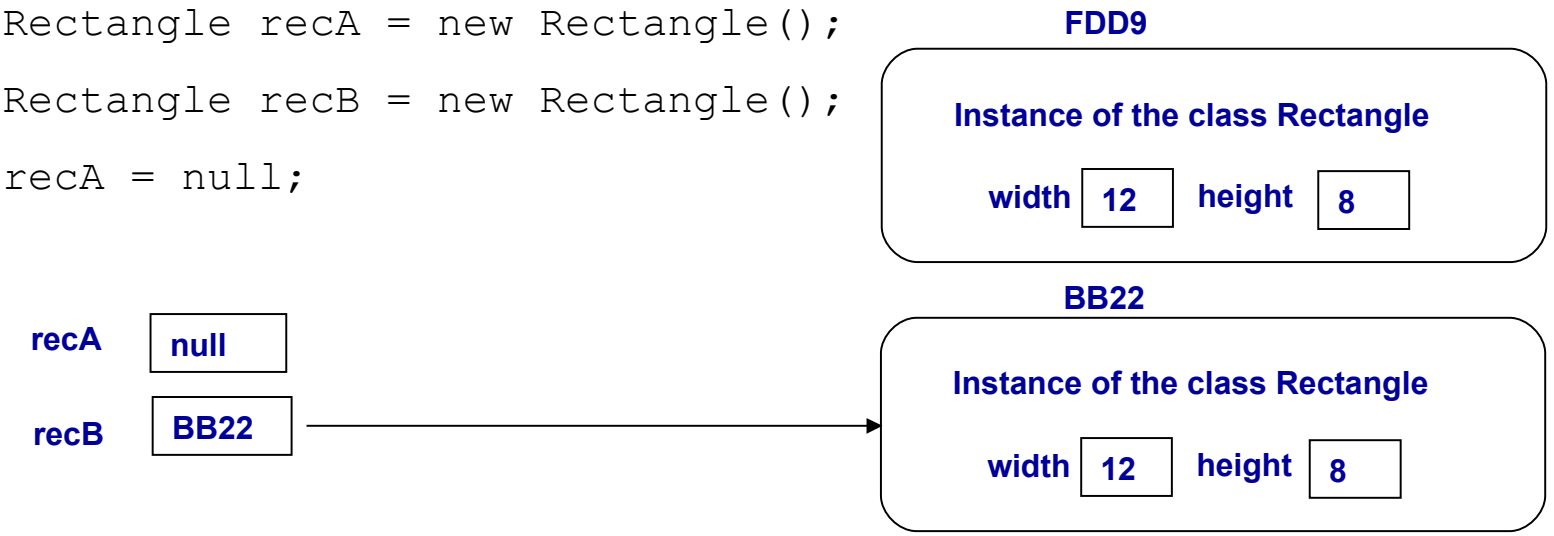
```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA.width = 12; recA.height = 8;
recB.width = 12; recB.height = 8;
if(recA.equals(recB)) {     …}
```



The `null` Value

- The special value `null` can be assigned to any class type variable.

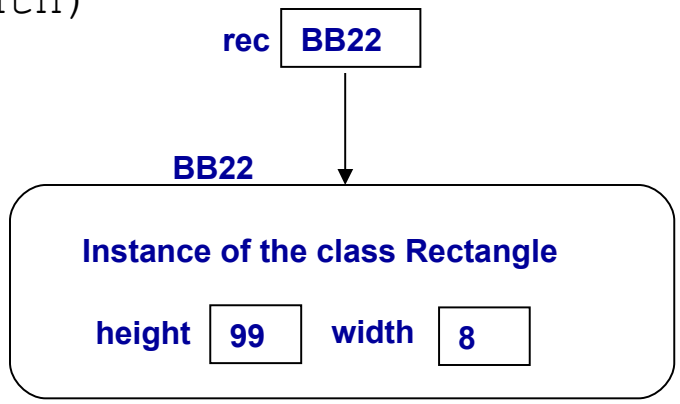
```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA = null;
```



The `this` Keyword

- The `this` keyword holds the reference of the current object. We can use it within instance methods or constructors only.

```
void setWidth(double width)
{
     this.width = width;
}
```



Fields Definition

```
[modifiers] <type> <name> [=default value];
```

Methods Definition

```
[modifiers] <return_type> <name> (
      <parameter_type> <parameter_name>, ..)
{
  <statements>
}
```

Methods Overloading

- The same method can be defined in the same class in several different versions.
- All versions should be defined with a returned value of the same type.
- The parameters list in each and every method should be with a different signature.

Constructors Definition

```
[modifiers] <class_name> (
      <parameter_type> <parameter name>, ..)
{
   <statements>
}
```

The Default Constructor

- The default constructor exists when we define a class without defining a specific constructor. The default constructor doesn't have parameters.

Constructors Overloading

- Constructors can be overloaded (like methods).

```
public Rectangle() {...}
public Rectangle(int num)  {...}
public Rectangle(int num1, int num2)  {...}
```

- Each and every constructor should be defined with a different signature.

Using `this` Within Constructors

- Placing the `this` keyword in the first line (within the constructor block) we can call another constructor.

Static Variables

- When we define a class variable together with the `static` modifier we shall get a static variable.
- The static variable is associated with the class as a whole rather than with a particular instance.

Static Variables

- The static variable can be accessed from within any method of the class.
- The static variable can also be accessed from outside of the class scope if its access modifier allows it.
- Static variables can be accessed by using a class type reference or by using the name of the class.

Static Methods

- When we define a method together with the `static` modifier we shall get a static method.
- The static method is associated with the class as a whole rather than with a particular instance.

Static Methods

- The static method can be called from within any method of the class.
- The static method can also be accessed from outside of the class scope if its access modifier allows it.
- Static methods can be called by using a class type reference or by using the name of the class.

Static Initializers

- A static block' is a block of code that doesn't belong to any specific method. The static block is prefixed with the `static` modifier.
- The static block contains code which is executed when the class is loaded to the JVM memory.
- The code within the static block is executed only once (when the class is loaded).

Final Variables

- A final variable can be set only once. Once the final variable was initialized it is impossible to assign it with another value.
- The final variable assignment can occur independently of its declaration.

Final Variables

- When having a final instance variable that its value isn't set together with its declaration, that final variable must be set in each and every constructor.
- When having a final static variable that its value isn't set together with its declaration, that final variable must be set in one of the static blocks.

The `package` Statement

- The package is a group of classes and interfaces.

  Each and every package can contain sub packages.

The `package` Statement

- The package statement starts with the keyword `package`.

```
package <package_name>.<package_name>….  ;
```

The following is an example for declaring a package with an hierarchy of three levels.

```
package com.zozobra.examples;
```

The `package` Statement

- Only one package statement per one source file is allowed.
- If the package statement isn't included within the source file then all classes in that specific source file will be part of the default package.

The `import` statement

- The import statement can import either a specific class or all classes (that belong to specific package).

```
import <package name>.<package name>.<class name>;
import <package name>.<package name>.*;
```

- The following example imports all classes that belong to the `java.awt` package.

```
import java.awt.*;
```