Singleton

could only allow one instance

```java
public class Singleton {

    private static Singleton firstInstance = null;

    private Singleton(){ }

    public static Singleton getInstance(){

        if(firstInstance == null){

            firstInstance = new Singleton();

        }

        return firstInstance;

    }
}
```

// what if we use thread?

synchronized → very slow

use synchronized block

Synchronized Blocks in Instance Methods

You do not have to synchronize a whole method. Sometimes it is preferable to synchronize only part of a method. Java synchronized blocks inside methods makes this possible.

Here is a synchronized block of Java code inside an unsynchronized Java method:

```
public void add(int value){

  synchronized(this){
     this.count += value;
  }
}
```

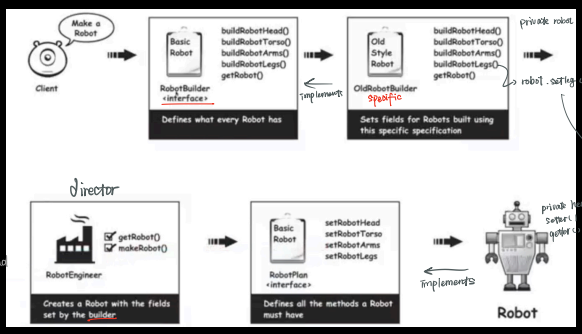
This example uses the Java synchronized block construct to mark a block of code as synchronized. This code will now execute as if it was a synchronized method.

Notice how the Java synchronized block construct takes an object in parentheses. In the example "this" is used, which is the instance the add method is called on. The object taken in the parentheses by the synchronized construct is called a monitor object. The code is said to be synchronized on the monitor object. A synchronized instance method uses the object it belongs to as monitor object.

What is the Singleton Pattern?

- It is used when you want to eliminate the option of instantiating more than one object
- I'll demonstrate it using a class that holds all the potential Scrabble letters and spits out new ones upon request
  - Each player will share the same potential letter list
  - Each player has their own set of letters

Builder




```
private robot Builder;
makeRobot {
  builder.makeHead
}
```

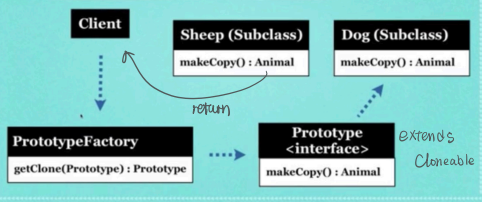
```java
public class TestRobotBuilder {

    public static void main(String[] args){

        RobotBuilder oldStyleRobot = new OldRobotBuilder();

        RobotEngineer robotEngineer = new RobotEngineer(oldStyleRobot);

        robotEngineer.makeRobot();

        Robot firstRobot = robotEngineer.getRobot();

    }
}
```

blueprint

What is the Builder Pattern?

- Pattern used to create objects made from a bunch of other objects
  - When you want to build an object made up from other objects
  - When you want the creation of these parts to be independent of the main object
  - Hide the creation of the parts from the client so both aren't dependent
  - The builder knows the specifics and nobody else does

Prototype Pattern UML Diagram



What is the Prototype Pattern?

- Creating new objects (instances) by cloning (copying) other objects
- Allows for adding of any subclass instance of a known super class at run time
- When there are numerous potential classes that you want to only use if needed at runtime
- Reduces the need for creating subclasses

```
public Animal makeCopy() {

    System.out.println("Sheep is Being Made");

    Sheep sheepObject = null;

    try {
        sheepObject = (Sheep) super.clone();
    } catch (CloneNotSupportedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return sheepObject;

}

public String toString(){

    return "Dolly is my Hero, Baaaaa";

}
```

```
1  public class TestCloning {
2
3      public static void main(String[] args){
4
5          CloneFactory animalMaker = new CloneFactory();
6
7          Sheep sally = new Sheep();
8
9          Sheep clonedSheep = (Sheep) animalMaker.getClone(sally);
10
11         System.out.println(sally);
12
13         System.out.println(clonedSheep);
14
15         System.out.println("Sally Hashcode: " + System.identityHashCode(System.ident
16
17         System.out.println("Clone Hashcode: " + System.identityHashCode(System.ident
18     }
19
20 }
```