Some creation patterns

Soliton & SimpleFactory

Singleton

- Ensure a class has only one instance and provide a global point of access to it.

```
class Referee{
    static Referee instance= null;
    private Referee() {
      String s = "";
    }
    public static Referee getReferee() {
      if (instance ==null) instance=new Referee();
      return instance;
    }
    public void whistle() {
      //...
    }
}
```

Singleton usage

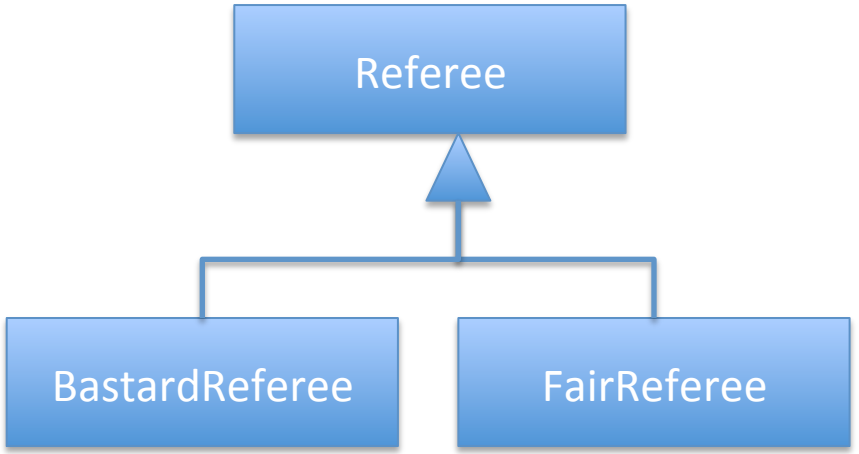
```
package myPackage;

public class Game{
      public static void main(String a[]) {
    new Game ();
  }

  Game () {
    //Referee a=new Referee (); // would give an error!
    Referee b=Referee.getReferee();
    Referee c=Referee.getReferee();
    System.out.println(b==c);
  }
}
```

SimpleFactory: isolate the code from the concrete implementating class

```
Referee x=new BastardReferee();
```




```
If (bastardnesslevel==0)
    Referee x=new FairReferee();
else
    Referee x=new BastardReferee();
```

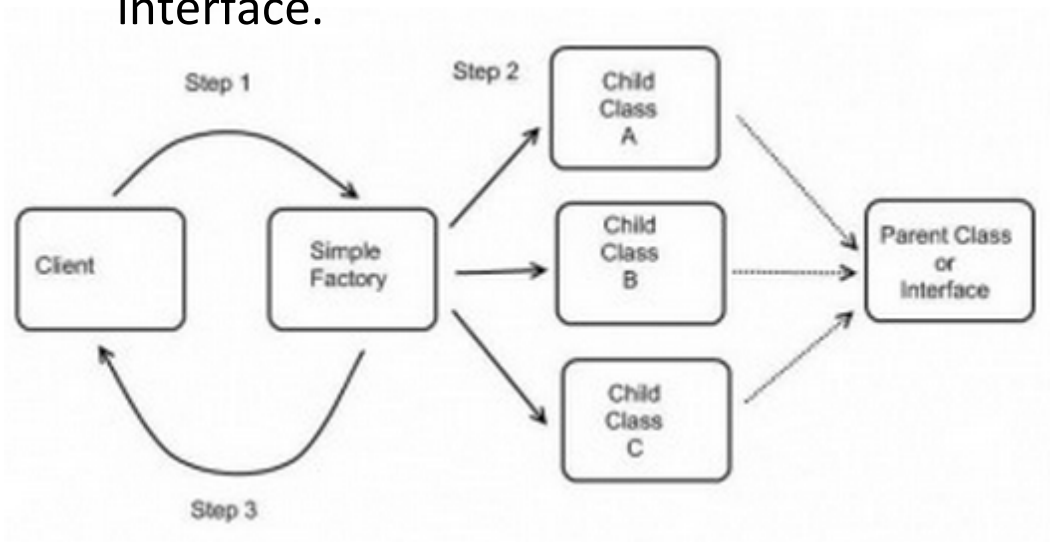
```
Referee x=RefereeFactory.getReferee(bastardnessLevel);
```

using a Simple Factory

1) you call a (possibly static) method in the factory. The call parameters tell the factory which class to create.

2) the factory creates your object. All the objects it can create either have the same parent class, or implement the same interface.

3) factory returns the object, the client expect is it to match the parent class / interface.




```
Parent x=Factory.create(p);


class Factory{
        static Parent create(Param p) {
                if (p…) return new ChildA();
                else return new ChildB();
        }
}
```

Factory

Factories are used to encapsulate instantiation.