

Prototype Design Pattern

- It is a type of Creational Design Pattern.
- It is used when you want to avoid multiple object creation of same instance, instead you copy the object to new object and then we can modify.
- Object which we are copying should provide copying feature by implementing cloneable interface.
- We can override clone() method to implement as per our need.

... then let's look at some
examples



Let us create first demo.java with main function.

```
package com.prototype;
```

```
public class Demo {
```

```
public static void main(String[] args) {
```

```
}
```

```
}
```

Now let us create the Book class where the book id and book name will be printed.

```
package com.prototype;
public class Book
{
    private int bid;
    private String bname;
    public int getBid() {
        return bid;
    }
    public void setBid(int bid) {
        this.bid = bid;
    }
    public String getBname() {
        return bname;
    }
    public void setBname(String bname) {
        this.bname = bname;
    }
    @Override
    public String toString() {
        return "Book [bid=" + bid + ", bname=" + bname + "]";
    }
}
```

Now let us consider all the Books will be stored in one BookShop, hence we are going to create a BookShop class.

```
package com.prototype;
import java.util.ArrayList;
import java.util.List;

public class BookShop
{
    private String shopName;
    List<Book> books = new ArrayList<>();
    public String getShopName() {
        return shopName;
    }
    public void setShopName(String shopName) {
        this.shopName = shopName;
    }
    public List<Book> getBooks() {
        return books;
    }
    public void setBooks(List<Book> books) {
        this.books = books;
    }
    @Override
    public String toString() {
        return "BookShop [shopName=" + shopName + ", books=" + books + "];"
    }
}
```

Now let us go to demo.java again.

```
package com.prototype;

public class Demo {
    public static void main(String[] args){
        BookShop bs = new BookShop(); //try to create object bs
        System.out.println(bs); // print will return null values
    }
}
```



**Will return an output like this because we have not specified any values in BookShop.
So create a method in BookShop which will add data to the BookShop.**

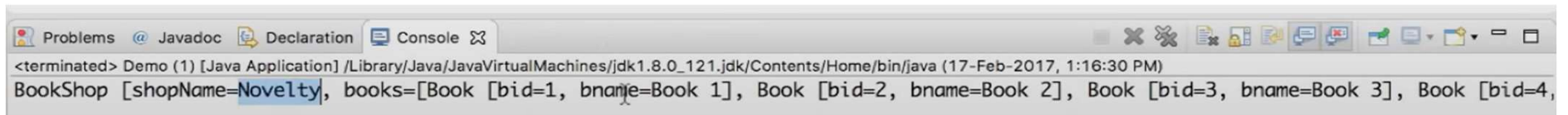
So let us come in BookShop again.

```
package com.prototype;
import java.util.ArrayList;
import java.util.List;
public class BookShop
{
    private String shopName;
    List<Book> books = new ArrayList<>();
    public String getShopName() {
        return shopName;
    }
    public void setShopName(String shopName) {
        this.shopName = shopName;
    }
    public List<Book> getBooks() {
        return books;
    }
    public void setBooks(List<Book> books) {
        this.books = books;
    }
    public void loadData(){
        for(int i=1; i<=10; i++){
            Book b = new Book();
            b.setBid(i);
            b.setBname("Books "+i);
            getBooks().add(b);
        }
    }
    @Override
    public String toString() {
        return "BookShop [shopName=" + shopName + ", books=" + books + "];"
    }
}
```


Then go to Demo again and add bookname and try to load the data from the DB.

```
package com.prototype;

public class Demo {
    public static void main(String[] args){
        BookShop bs = new BookShop(); //try to create object bs
        bs.setShopName("Shop1");
        bs.loadData();
        System.out.println(bs);
    }
}
```



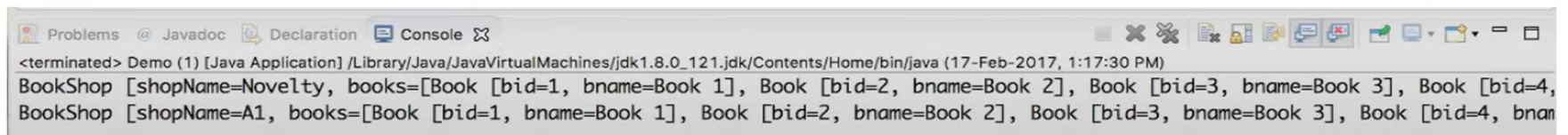
Will return an output like this after loading the data from BookShop. (ShopName is modified...)

Up to this, there is no Prototype Design Pattern.

So lets do the following in Demo.java again...

```
package com.prototype;
public class Demo {
public static void main(String[] args){
    BookShop bs = new BookShop(); //try to create object bs
    bs.setShopName("Shop1");
    bs.loadData();
    System.out.println(bs);

    BookShop bs1 = new BookShop(); //try to create object bs
    bs1.setShopName("Shop2");
    bs1.loadData();
    System.out.println(bs1);
}
}
```



The screenshot shows a Java IDE console window with the following output:

```
<terminated> Demo (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (17-Feb-2017, 1:17:30 PM)
BookShop [shopName=Novelty, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=3, bname=Book 3], Book [bid=4, bname=Book 4]]]
BookShop [shopName=A1, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=3, bname=Book 3], Book [bid=4, bname=Book 4]]]
```

- **So the second object is loading the same data from the database again.**
- **But if it is required to create different object to access the same data then second object also can try to copy or clone the same from the first object.**
- **That is the concept of Prototype Design Method.**

So lets do the following in Demo.java again...

```
package com.prototype;
public class Demo {
public static void main(String[] args){
    BookShop bs = new BookShop(); //try to create object bs
    bs.setShopName("Shop1");
    bs.loadData();
    System.out.println(bs);

    BookShop bs1 = bs.clone(); //instead of creating new object create clone
    bs1.setShopName("Shop2");
    bs1.loadData();
    System.out.println(bs1);
}
}
```

‘Clone’ method is in the object class. However as it is protected there so to use that it requires to set the permission.

```
package com.prototype;
import java.util.ArrayList;
import java.util.List;
public class BookShop implements Cloneable
```

```

//////////////////////////////////// next slide
}

```

- `//to achieve cloning we have to override a clone method`

```
@Override
protected Object clone() throws CloneNotSupportedException {
    // TODO Auto-generated method stub
    return super.clone();
}
```

And in Demo.java the following:

```
BookShop bs1 = (BookShop) bs.clone();
```

And

```
public static void main(String[] args) throws CloneNotSupportedException
```

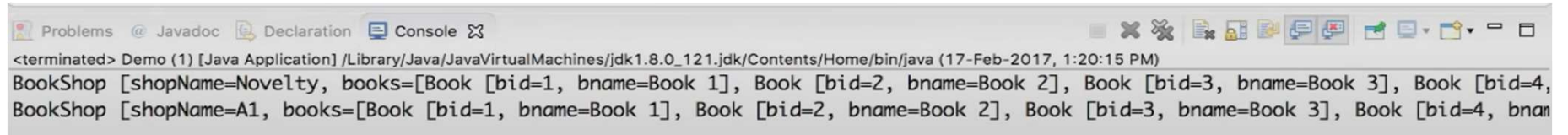
Just like...

```
package com.prototype;

public class Demo {

    public static void main(String[] args) throws CloneNotSupportedException
    {
        BookShop bs = new BookShop();
        bs.setShopName("Shop1");
        bs.loadData();
        System.out.println(bs);

        BookShop bs1 = (BookShop)bs.clone();
        bs1.setShopName("Shop2");
        System.out.println(bs1);
    }
}
```



```
Problems @ Javadoc Declaration Console
<terminated> Demo (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (17-Feb-2017, 1:20:15 PM)
BookShop [shopName=Novelty, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=3, bname=Book 3], Book [bid=4,
BookShop [shopName=A1, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=3, bname=Book 3], Book [bid=4, bnan
```

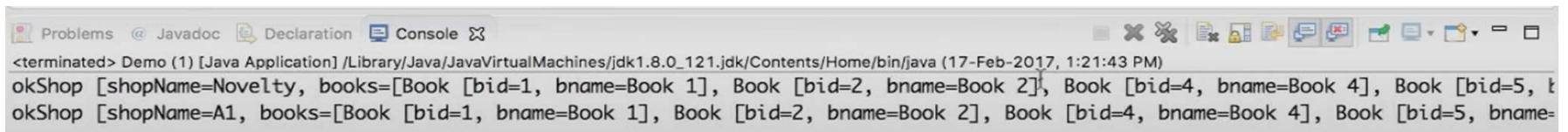
But one problem...

```
package com.prototype;

public class Demo {

    public static void main(String[] args) throws CloneNotSupportedException
    {
        BookShop bs = new BookShop();
        bs.setShopName("Shop1");
        bs.loadData();
        bs.getBooks().remove(2);
        System.out.println(bs);

        BookShop bs1 = (BookShop)bs.clone();
        bs1.setShopName("Shop2");
        System.out.println(bs1);
    }
}
```



```
<terminated> Demo (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (17-Feb-2017, 1:21:43 PM)
okShop [shopName=Novelty, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=4, bname=Book 4], Book [bid=5, bname=Book 5]]]
okShop [shopName=A1, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=4, bname=Book 4], Book [bid=5, bname=Book 5]]]
```


- **So results deleting from both bs and bs1.**
- **Instead of creating two separate object, by cloning we are getting two references of a single object.**
- **Thus this kind of cloning is not creating too much impact.**
- **So how to achieve ‘Deep Cloning’...**
- **Go to ‘BookShop.java’ and do the following...**

```
package com.prototype;
import java.util.ArrayList;
import java.util.List;
public class BookShop implements Cloneable
```

```

//////////////////////////////////// next slide
}

```

```
@Override
protected BookShop clone() throws CloneNotSupportedException {
// TODO Auto-generated method stub
return super.clone();
}
```

// And in Demo.java, in case of deepcloning in Bookshop, when you are returning bookshop and not using object then you dont need to cast bookshop :

```
BookShop bs1 = bs.clone();
```

And remove the **return super.clone()** also, like the following:

```
protected BookShop clone() throws CloneNotSupportedException {
// TODO Auto-generated method stub
//return super.clone();
BookShop shop = new BookShop();
for(Book b : this.getBooks()) // Don't put these three rows first and with return only
{
    // no data will be shown in output for the second object as nothing is returned
    shop.getBooks().add(b);
}
return shop;
}
```

Now finally go to Demo.java and complete it...

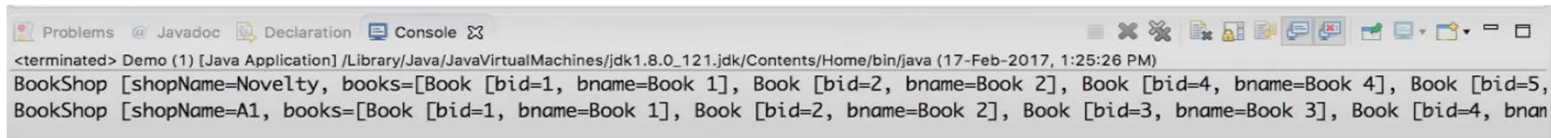
```
package com.prototype;

public class Demo {

public static void main(String[] args) throws CloneNotSupportedException
{
BookShop bs = new BookShop();
bs.setShopName("Shop1");
bs.loadData();

BookShop bs1 = bs.clone();
bs.getBooks().remove(2);
bs1.setShopName("Shop2");

System.out.println(bs);
System.out.println(bs1);
}
}
```



The screenshot shows a Java IDE console window with the following output:

```
<terminated> Demo (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (17-Feb-2017, 1:25:26 PM)
BookShop [shopName=Novelty, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=4, bname=Book 4], Book [bid=5,
BookShop [shopName=A1, books=[Book [bid=1, bname=Book 1], Book [bid=2, bname=Book 2], Book [bid=3, bname=Book 3], Book [bid=4, bnan
```