Now, in AbstractFactoryExample1, we use the definition provided in the book.

Now, the definition is in java. And, it is the following:

**/\*abstract base product class\*/**public abstract class Computer

{

public abstract String getRAM();

public abstract String getHDD();

public abstract String getCPU();

//we make these methods abstract so that the abstract class does not need to define them. And, the normal class extends abstract class must define them

@Override

public String toString()

{

return "RAM= "+this.getRAM()+", HDD="+this.getHDD()+", CPU="+this.getCPU();

}

}

/\*Concrete product class\*/

public class PC extends Computer

{

private String ram;

private String hdd;

private String cpu;

public PC(String ram, String hdd, String cpu)

{

this.ram=ram;

this.hdd=hdd;

this.cpu=cpu;

}

@Override

public String getRAM()

{

return this.ram;

}

@Override

public String getHDD()

{

return this.hdd;

}

/\*class Server\*/

public class Server extends Computer {

private String ram;

private String hdd;

private String cpu;

public Server(String ram, String hdd, String cpu){

this.ram=ram;

this.hdd=hdd;

this.cpu=cpu;

}

@Override

public String getRAM() {

return this.ram;

}

@Override

public String getHDD() {

return this.hdd;

}

@Override

public String getCPU() {

return this.cpu;

}

}

**Now, as you will see, AbstractFactoryExample1 is the obvious choice.**

Now, I try to improvise. Because, the abstract class is always different than interface. You can have members and you can reuse them (those members need not to be public static final) . So, I made some changes. That improvised version is AbstractFactoryExample2.

Now, I tried to improvise further in AbstractFactoryExample3. Using the fact, that a pure virtual function can have a body, which will not be inherited. But, you can use the definition using scope operator.

Now, which design is actually better. Which one has high flaws, I need to know. **So, currently, I am seeking deeper knowledge.**