**Exercise 1: Implementing the Singleton Pattern (Mandatory)**

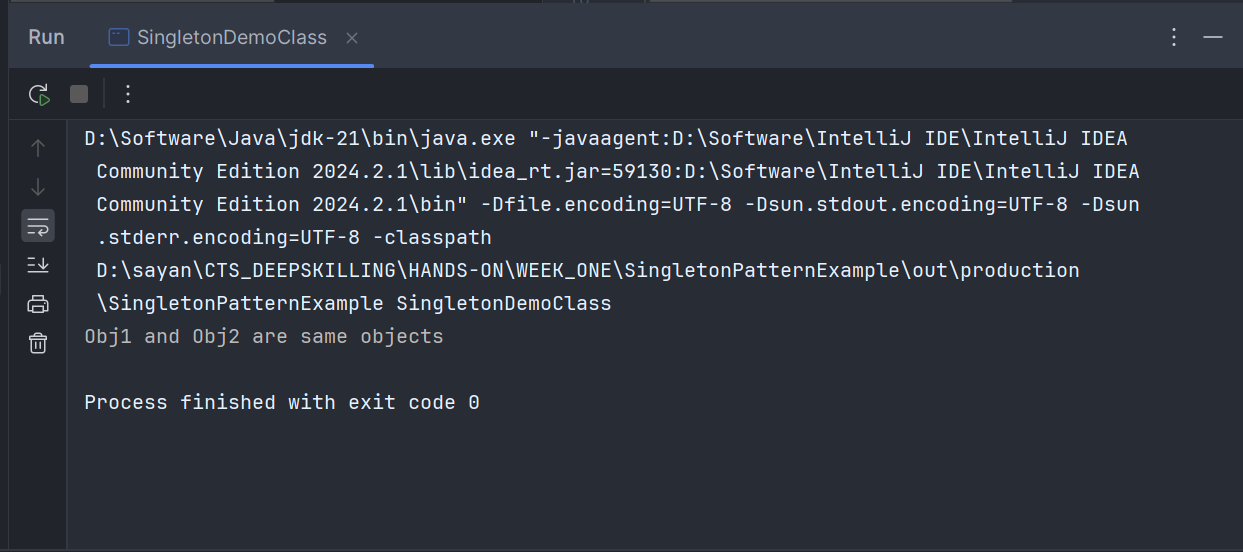
**PROGRAM:**

*//test class*

*public class* SingletonDemoClass {  
 *public static void* main(String[] args) {  
  
 *//creating two objects to check whether they are same or not* Logger obj1 = Logger.getInstance();  
 Logger obj2 = Logger.getInstance();  
  
 *//using == to use object level comparison  
 if*(obj1 == obj2){  
 System.out.println("Obj1 and Obj2 are same objects");  
 }  
 *else*{  
 System.out.println("Obj1 and Obj2 are different objects");  
 }  
  
 }  
}

//Logger class  
*class* Logger  
{  
 *//private instance of Logger  
 private static* Logger instance = *null*;  
  
 *//private constructor  
 private* Logger(){}  
  
 *//static method to return instance  
 public static* Logger getInstance(){  
 *if*(instance == *null*){  
 instance = *new* Logger();  
 }  
  
 *return* instance;  
 }  
}

**OUTPUT:**

****

**Exercise 2: Implementing the Factory Method Pattern (Mandatory)**

**PROGRAM:**

*//interface for WordDocument, PdfDocument, and ExcelDocument classes*

*public interface* Document {  
 *public void* openDocument();  
}

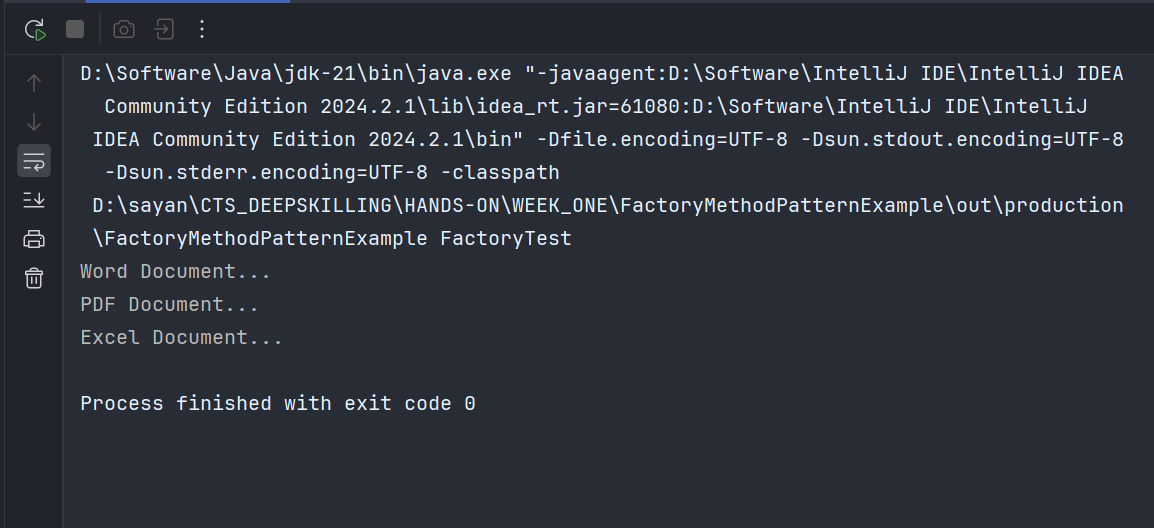
*//abstract class for factory classes*

*public abstract class* DocumentFactory {  
 *public abstract* Document createDocument();  
}

*//test class*

*public class* FactoryTest {  
 *public static void* main(String[] args) {  
 *//word factory* DocumentFactory wordFactory = *new* WordDocumentFactory();  
 Document wordFile = wordFactory.createDocument();  
 wordFile.openDocument();  
  
 *//pdf factory* DocumentFactory pdfFactory = *new* PdfDocumentFactory();  
 Document pdfFile = pdfFactory.createDocument();  
 pdfFile.openDocument();  
  
 *//excel factory* DocumentFactory excelFactory = *new* ExcelDocumentFactory();  
 Document excelFile = excelFactory.createDocument();  
 excelFile.openDocument();  
 }  
}  
  
*//Document classes  
  
class* WordDocument *implements* Document{  
 *@Override  
 public void* openDocument() {  
 System.out.println("Word Document...");  
 }  
}  
  
*class* PdfDocument *implements* Document{  
 *@Override  
 public void* openDocument() {  
 System.out.println("PDF Document...");  
 }  
}  
  
*class* ExcelDocument *implements* Document{  
 *@Override  
 public void* openDocument() {  
 System.out.println("Excel Document...");  
 }  
}  
  
*//Factory classes  
  
class* WordDocumentFactory *extends* DocumentFactory{  
 *@Override  
 public* Document createDocument() {  
 *return new* WordDocument();  
 }  
}  
  
*class* PdfDocumentFactory *extends* DocumentFactory{  
 *@Override  
 public* Document createDocument() {  
 *return new* PdfDocument();  
 }  
}  
  
*class* ExcelDocumentFactory *extends* DocumentFactory{  
 *@Override  
 public* Document createDocument() {  
 *return new* ExcelDocument();  
 }  
}

**OUTPUT:**

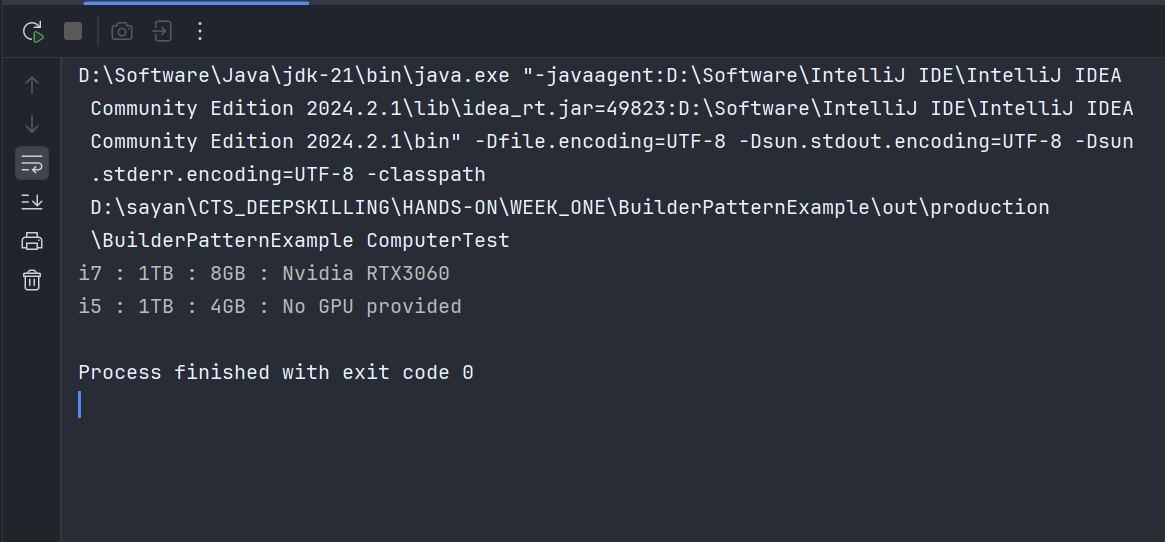
****

**Exercise 3: Implementing the Builder Pattern (Optional)**

**PROGRAM:**

*//test class  
public class* ComputerTest {  
 *public static void* main(String[] args) {  
  
 *//building with all optional and required components* Computer pc = *new* Computer.ComputerBuilder("i7","8GB","1TB").setGpu("Nvidia RTX3060").build();  
 System.out.println(pc.getCpu()+" : "+pc.getStorage()+" : "+pc.getRam()+ " : "+pc.getGpu());  
  
 *//building with only required components* Computer homePc = *new* Computer.ComputerBuilder("i5","4GB","1TB").build();  
 System.out.println(homePc.getCpu()+" : "+homePc.getStorage()+" : "+homePc.getRam()+ " : "+homePc.getGpu());  
 }  
}  
  
*//class  
class* Computer{  
  
 *//required components  
 private* String cpu;  
 *private* String ram;  
 *private* String storage;  
  
 *//optional components  
 private* String gpu;  
  
 *//private constructor  
 private* Computer(ComputerBuilder builder){  
 *this*.cpu = builder.cpu;  
 *this*.ram = builder.ram;  
 *this*.storage = builder.storage;  
 *this*.gpu = builder.gpu;  
 }  
  
 *//getters  
 public* String getCpu(){  
 *return this*.cpu;  
 }  
  
 *public* String getRam(){  
 *return this*.ram;  
 }  
  
 *public* String getStorage(){  
 *return this*.storage;  
 }  
  
 *public* String getGpu(){  
 *if*(*this*.gpu == *null*)  
 *return* "No GPU provided";  
 *else  
 return this*.gpu;  
 }  
  
 *//static builder class  
 public static class* ComputerBuilder{  
 *//required components  
 private* String cpu;  
 *private* String ram;  
 *private* String storage;  
  
 *//optional components  
 private* String gpu;  
  
 *//constructor to initialize required components* ComputerBuilder(String cpu,String ram,String storage){  
 *this*.cpu = cpu;  
 *this*.ram = ram;  
 *this*.storage = storage;  
 }  
  
 *//method to initialize optional components  
 public* ComputerBuilder setGpu(String gpu){  
 *this*.gpu = gpu;  
 *return this*;  
 }  
  
 *public* Computer build(){  
 *return new* Computer(*this*);  
 }  
  
  
 }  
  
}

**OUTPUT:**

****