**Introduction to Software Engineering**

**Final Exam**

**Time: 1 hour 15 minutes**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_Higgins\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_Timothy\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** (Print Name) (Last Name) (First Name)

**INSTRUCTIONS FOR STUDENTS:**

• No portable devices (e.g., phone, tablet, laptop, etc.) are allowed during the examination at any time. You will be dismissed from the testing site if you are seen with those portable devices above. • Please do not talk during the examination.

• If you have a problem please raise your hand and wait quietly for an instructor.

• Write your name clearly in the space above when directed to do so.

• At the conclusion of your examination please refrain from speaking until you are outside the exam room as there may still be other examinations in progress.

• You may use the following:

o Cheat sheets maximum five pages (a page is one side of a sheet of paper).

**I attest that I spent no more than 75 minutes to complete the exam. I used only the materials described above. I did not receive assistance from anyone during the taking of this exam.**

**Student's Signature \_\_Timothy Higgins\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1. Explain qualifiers in the class diagram by using code examples. (6 points)**

**2. When to use the composite design pattern in object-oriented programming? (6 points)**

**3. Create the class diagram with generalization relationship regarding the implementation below. (6 points)**

public class User { /\* ... \*/ }

public class RegisteredUser extends User { /\* ... \*/ }

public class AnonymousUser extends User { /\* ... \*/ }

**4. Create the class diagram with aggregation or composition relationship regarding the implementation below. (6 points)**

class SessionEventListenerManagerImpl {

private List<SessionEventListener> listenerList;

public SessionEventListenerManagerImpl(SessionEventListener listener) {

if (listenerList == null) {

listenerList = new ArrayList<SessionEventListener>();

}

listenerList.add(listener);

}

}

class SessionEventListener {

public void transactionCompletion(boolean successful) {

// Implementation goes here.

}

}

**5.** It is impossible to represent all relevant information about a system’s architecture in a single diagram. It might show how a system is decomposed into modules, how the runtime processes interact, or the different ways in which system components are distributed across a network. Because all of these are useful at different times, for both design and documentation, you usually need to present multiple views of the software architecture. Software architectures are typically documented from several different perspectives or views. **Possible views include** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**(6 points)**

**6. Complete the following program that uses Model-View-Controller (MVC) pattern. (6 points)**

**public class** MVCProgram {

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

SystemInfoModel model = **new** SystemInfoModel();

SystemInfoViewer view = **new** SystemInfoViewer(model.getSystemInfo());

SystemInfoUIController controller = **new** SystemInfoUIController(model, view); controller.control();

view.redraw();

}

}

**class** SystemInfoModel {

StringBuilder info;

**public** SystemInfoModel() {

info = **new** StringBuilder();

}

**void** setSystemInfo() {

info.setLength(0);

/\* Total number of processors or cores available to the JVM \*/

info.append("Available processors (cores): "

+ Runtime.*getRuntime*().availableProcessors()); /\* Total amount of free memory available to the JVM \*/

info.append(", Free memory (bytes): " + Runtime.*getRuntime*().freeMemory()); /\* This will return Long.MAX\_VALUE if there is no preset limit \*/

**long** maxMemory = Runtime.*getRuntime*().maxMemory();

/\* Maximum amount of memory the JVM will attempt to use \*/

info.append(", Maximum memory (bytes): "

+ (maxMemory == Long.***MAX\_VALUE*** ? "no limit" : maxMemory)); /\* Total memory currently available to the JVM \*/

info.append(", Total memory available to JVM (bytes): "

+ Runtime.*getRuntime*().totalMemory());

}

String getSystemInfo() {

**return** info.toString();

}

}

**class** SystemInfoViewer {

**private static** Display *display*;

**private static** Shell *shell*;

**private** Button button;

**private** Text textfield;

**public** SystemInfoViewer(String str) {

*display* = **new** Display();

*shell* = **new** Shell(*display*);

*shell*.setSize(500, 200);

*shell*.setText("MVC Example");

*shell*.setLayout(**new** RowLayout());

button = **new** Button(*shell*, SWT.***PUSH***);

button.setText("Get System Information:");

textfield = **new** Text(*shell*, SWT.***SHADOW\_IN***);

textfield.setSize(1000, 50);

textfield.setText(str);

}

**public** Button getButton() {

**return this**.button;

}

**public void** setTextField(String str) {

textfield.setText(str);

}

**public** Text getTextfield() {

**return this**.textfield;

}

**public void** redraw() {

*shell*.pack();

*shell*.open();

**while** (!*shell*.isDisposed()) {

**if** (!*display*.readAndDispatch())

*display*.sleep();

}

*display*.dispose();

}

}

**class** SystemInfoUIController {

**private** SystemInfoModel model;

**private** SystemInfoViewer view;

**public** SystemInfoUIController(SystemInfoModel model, SystemInfoViewer view) { // (1) Write your answer here

}

**public void** control() {

view.getButton().addSelectionListener(**new** SelectionListener() {

**public void** widgetSelected(SelectionEvent event) {

linkBtnAndTextField();

}

**public void** widgetDefaultSelected(SelectionEvent event) {

}

});

}

**private void** linkBtnAndTextField() {

model.setSystemInfo();

// (2) Write your answer here

view.redraw();

}

}

**7. Complete the following program that uses Java Native Interface (JNI) allowing a Java program to access to a shared library written in C programming language. (6 points)**

**/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* MyApp.java**

**\*/**

public class MyApp {

public static void main(String[] args) throws Exception {

CameraDeviceDriverJNI.savePhoto();

}

}

**/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* CameraDeviceDriverJNI.java**

**\*/**

public class CameraDeviceDriverJNI {

static {

// (1) Write your answer here

System.loadLibrary(" ");

}

// (2) Write your answer here

native static void ;

}

**/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* CameraDeviceDriverJNI.h**

**\*/**

#include <jni.h>

#ifndef \_Included\_CameraDeviceDriverJNI

#define \_Included\_CameraDeviceDriverJNI

#ifdef \_\_cplusplus

extern "C" {

#endif

JNIEXPORT void JNICALL Java\_CameraDeviceDriverJNI\_savePhoto(JNIEnv \*, jclass); #ifdef \_\_cplusplus

}

#endif

#endif

**/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* CameraDeviceDriverJNI.c**

**\*/**

#include "CameraDeviceDriverJNI.h"

#include "camera\_device\_driver.h"

#include <unistd.h>

JNIEXPORT void JNICALL Java\_CameraDeviceDriverJNI\_savePhoto (JNIEnv \* env, jclass clazz) { save\_photo();

}

**/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* camera\_device\_driver.h**

**\*/**

void save\_photo();

**/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* camera\_device\_driver.c**

**\*/**

#include <stdio.h>

#include "camera\_device\_driver.h"

void save\_photo() {

printf("save photo\n");

}

**/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Makefile**

**\*/**

CFLAGS=-I$(JAVA\_HOME)/include/ -I$(JAVA\_HOME)/include/linux/

CPRG=CameraDeviceDriverJNI

JAVAPRG=CameraDeviceDriverJNI.java MyApp.java

all: buildlinux

javabuild:

javac $(JAVAPRG)

buildlinux: javabuild

gcc -fPIC $(CFLAGS) -shared -o lib$(CPRG).so $(CPRG).c camera\_device\_driver.c

clean:

rm -f \*.class $(CPRG) \*.o \*.so

**8. Remote Method Invocation (RMI) performs the object-oriented equivalent of Remote Procedure Call (RPC), with support for direct transfer of serialized Java classes and distributed garbage collection. Complete the implementation of the RMI client/server program. (6 points)**

**public class** Customer **implements** Serializable {

String name, address;

**public** Customer(String name, String address) {

**this**.name = name;

**this**.address = address;

}

**public** String getName() {

**return** name;

}

**public void** setName(String name) {

**this**.name = name;

}

**public** String getAddress() {

**return** address;

}

**public void** setAddress(String address) {

**this**.address = address;

}

}

**public class** RmiClient {

**public static void** main(String args[]) **throws** Exception {

(**new** Thread(**new** Client1())).start();

(**new** Thread(**new** Client2())).start();

}

}

**class** Client1 **implements** Runnable {

**public void** run() {

Customer customer = **new** Customer("customer1", "address1");

RmiServerIntf obj = (RmiServerIntf) Naming.*lookup*("//137.48.186.XYZ/RmiServer"); obj.setCustomer(customer);

}

}

**class** Client2 **implements** Runnable {

**public void** run() {

RmiServerIntf obj;

obj = (RmiServerIntf) Naming.*lookup*("//137.48.186.XYZ/RmiServer");

Customer customer = **null**;

**while** (**true**) {

**if** (customer == **null**) {

customer = obj.getCustomer();

Thread.*sleep*(100);

**continue**;

} **else** {

System.***out***.println(customer.getName());

System.***out***.println(customer.getAddress());

**break**;

}

}

}

}

**public class** RmiServer **extends** UnicastRemoteObject **implements** RmiServerIntf { **private static final long *serialVersionUID*** = 1L;

**private** Customer customer;

**public static void** main(String args[]) **throws** Exception {

LocateRegistry.*createRegistry*(1099);

RmiServer obj = **new** RmiServer();

Naming.*rebind*("//localhost/RmiServer", obj);

}

**public** RmiServer() **throws** RemoteException {

**super**(0);

}

@Override

**public void** setCustomer(Customer customer) **throws** RemoteException { **this**.customer = customer;

}

@Override

**public** Customer getCustomer() **throws** RemoteException {

**return this**.customer;

}

}

public interface RmiServerIntf extends Remote {

// (1) Write your answer here.

public throws RemoteException;

// (2) Write your answer here.

public throws RemoteException; }

**9. As patterns have made a huge impact on object-oriented software design, they have become a vocabulary for talking about software design. Describe why software design patterns are important in object-oriented programming. (6 points)**

**10. The observer design pattern maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. Complete the following program that uses the observer design pattern. (18 points)**

public class MainProgram {

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

ConcreteSubjectModel subjectModel = new ConcreteSubjectModel(0);

SouthView viewOne = new SouthView(subjectModel);

NorthView viewTwo = new NorthView(subjectModel);

Controller controller = new Controller(subjectModel,

viewOne, viewTwo);

controller.control();

}

});

}

}

class Controller {

ConcreteSubjectModel subjectModel;

SouthView viewOne;

NorthView viewTwo;

public Controller(ConcreteSubjectModel model, SouthView viewOne, NorthView viewTwo) { this.subjectModel = model;

this.viewOne = viewOne;

this.viewTwo = viewTwo;

}

public void control() {

ActionListener actionListener = new ActionListener() {

public void actionPerformed(ActionEvent actionEvent) {

linkBtnAndLabel();

}

};

viewOne.getButton().addActionListener(actionListener);

viewTwo.getButton().addActionListener(actionListener);

}

public void linkBtnAndLabel() {

subjectModel.incX();

viewOne.setText(subjectModel.getX());

viewTwo.setText(subjectModel.getX());

}

}

class SubjectModel {

List<Observer> views = new ArrayList<Observer>();

public void addObserver(Observer o) {

views.add(o);

}

public void deleteObserver(Observer o) {

views.remove(o);

}

public void notifyAllViews() {

for (int i = 0; i < views.size(); i++) {

// (1) Write your answer here

}

}

}

class ConcreteSubjectModel extends SubjectModel { int x;

public ConcreteSubjectModel(int x) {

this.x = x;

}

public void incX() {

x++;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

// (2) Write your answer here

}

}

abstract class Observer {

ConcreteSubjectModel subjectModel;

JFrame frame;

JLabel label;

JButton button;

public Observer(String title) {

frame = new JFrame(title);

frame.getContentPane().setLayout(new BorderLayout());

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(200, 200);

frame.setVisible(true);

label = new JLabel("-");

frame.getContentPane().add(label, BorderLayout.CENTER);

button = new JButton("Button");

}

abstract void update();

}

class SouthView extends Observer {

public SouthView(ConcreteSubjectModel subject) {

super("South View");

this.label.setFont(new Font("Courier New", Font.ITALIC, 100)); frame.getContentPane().add(button, BorderLayout.SOUTH);

this.subjectModel = subject;

// (3) Write your answer here

}

public JButton getButton() {

return button;

}

public void setText(int text) {

// (4) Write your answer here

}

@Override

void update() {

label.setText("" + this.subjectModel.getX());

}

}

class NorthView extends Observer {

public NorthView(ConcreteSubjectModel subject) {

super("North View");

label.setFont(new Font("Serif", Font.PLAIN, 100));

frame.getContentPane().add(button, BorderLayout.NORTH);

this.subjectModel = subject;

// (5) Write your answer here

}

public JButton getButton() {

return button;

}

public void setText(int text) {

// (6) Write your answer here

}

@Override

void update() {

label.setText("" + subjectModel.getX());

}

}

**11. The composite pattern describes that a group of objects is to be treated in the same way as a single instance of an object. The intent of a composite is to “compose” objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly. Complete the following program that uses the composite design pattern. (6 points)**

abstract class NetworkInfo {

public abstract void sendAttribute();

public abstract void sendMessage();

}

class GUIComposite extends NetworkInfo {

private List<NetworkInfo> childComponents = new ArrayList<NetworkInfo>(); public void sendAttribute() {

for (NetworkInfo info : childComponents) {

// (1) Write your answer here

}

}

public void sendMessage() {

for (NetworkInfo info : childComponents) {

// (2) Write your answer here

}

}

public void add(NetworkInfo obj) {

childComponents.add(obj);

}

public void remove(NetworkInfo obj) {

childComponents.remove(obj);

}

}

class NetworkInfoEvent extends NetworkInfo {

private String attr, msg;

public NetworkInfoEvent(String attr, String msg) {

this.attr = attr;

this.msg = msg;

}

public void sendAttribute() {

System.out.println("Send " + this.attr + " from NetworkInfoEvent");

}

public void sendMessage() {

System.out.println("Send " + this.msg + " from NetworkInfoEvent");

}

}

class NetworkInfoRequest extends NetworkInfo {

private String attr;

private String msg;

public NetworkInfoRequest(String attr, String msg) {

this.attr = attr;

this.msg = msg;

}

public void sendAttribute() {

System.out.println("Send " + this.attr + " from NetworkInfoRequest"); }

public void sendMessage() {

System.out.println("Send " + this.msg + " from NetworkInfoRequest"); }

}

class NetworkInfoStatus extends NetworkInfo {

private String attr;

private String msg;

public NetworkInfoStatus(String attr, String msg) {

this.attr = attr;

this.msg = msg;

}

public void sendAttribute() {

System.out.println("Send " + this.attr + " from NetworkInfoSatus");

}

public void sendMessage() {

System.out.println("Send " + this.msg + " from NetworkInfoSatus");

}

}

public class Program {

public static void main(String[] args) {

NetworkInfoEvent event = new NetworkInfoEvent("evt:attr", "evt:msg"); NetworkInfoRequest request = new NetworkInfoRequest("req:attr", "req:msg"); NetworkInfoStatus status = new NetworkInfoStatus("sta:attr", "sta:msg"); GUIComposite composite = new GUIComposite();

composite.add(event);

composite.add(request);

composite.add(status);

composite.sendAttribute();

composite.sendMessage();

}

}

**12. A developer would like to implement a client program using the *WebBrower* component that should reuse a plugin component called *MSWordPlugin*. The *MSWordPlugin* component has been developed but it is not compatible with the *WebBrowser* component. This incongruity can be reconciled by changing an additional level of indirection, such as *WebBrowser* and *WebBrowserAdapter* classes. Please help the developer update the following program using the Object Adapter Pattern. (6 points)**

**public class** Program {

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

IWebBrowser webBrowser = **new** WebBrowser();

webBrowser.read("mypdf.pdf");

webBrowser.read("mymsword.docx");

}

}

**interface** IWebBrowser {

**public abstract void** read(String fileName);

}

**class** WebBrowser **implements** IWebBrowser {

WebBrowserAdapter documentAdapter;

**public void** read(String fileName) {

**if** (fileName.endsWith(".pdf") || fileName.endsWith(".docx")) {

documentAdapter = **new** WebBrowserAdapter(fileName);

documentAdapter.read(fileName);

} **else** {

System.*out*.println("Invalid file: " + fileName);

}

}

}

**class** WebBrowserAdapter **implements** IWebBrowser {

WebBrowserPlugin plugin;

**public** WebBrowserAdapter(String fileName) {

**if** (fileName.endsWith(".pdf")) {

plugin = **new** PDFPlugin();

}

**else if** (fileName.endsWith(".docx")) {

// (1) Write your answer here

}

}

**public void** read(String fileName) {

**if** (fileName.endsWith(".pdf")) {

plugin.read(fileName);

}

**else if** (fileName.endsWith(".docx")) {

// (2) Write your answer here

}

}

}

**abstract class** WebBrowserPlugin {

**public abstract void** read(String fileName);

}

**class** PDFPlugin **extends** WebBrowserPlugin {

**public void** read(String fileName) {

System.*out*.println("Reading PDF file: " + fileName);

}

}

**class** MSWordPlugin **extends** WebBrowserPlugin {

**public void** read(String fileName) {

System.*out*.println("Reading MSWord file: " + fileName); }

}

**13. There are two types of thread schedulers, such as ThreadScheduler\_A and ThreadScheduler\_B, and three types of operating systems, such as OS1, OS2, and OS3. Intuitively, the number of classes we would have to define is the product of the number of scheduling schemes and the number of operating systems. The bridge design pattern can be applied for refactoring this exponentially explosive inheritance hierarchy into two orthogonal hierarchies – one for platform-independent abstractions, and the other for platform-dependent implementations. (6 points)**

|  |
| --- |
| Original Version |

abstract class ThreadScheduler {

public abstract void schedule();

}

abstract class ThreadScheduler\_A extends ThreadScheduler {

public void schedule() {

System.*out*.println("\tA approach");

}

public abstract void scheduleThreadsWithTasks();

}

class OS1\_A extends ThreadScheduler\_A {

public void scheduleThreadsWithTasks() {

schedule();

System.*out*.println("\t\tSchedule threads with tasks in OS1.");

}

}

class OS2\_A extends ThreadScheduler\_A {

public void scheduleThreadsWithTasks() {

schedule();

System.*out*.println("\t\tSchedule threads with tasks in OS2.");

}

}

class OS3\_A extends ThreadScheduler\_A {

public void scheduleThreadsWithTasks() {

schedule();

System.*out*.println("\t\tSchedule threads with tasks in OS3.");

}

}

abstract class ThreadScheduler\_B extends ThreadScheduler {

public void schedule() {

System.*out*.println("\tB approach");

}

public abstract void scheduleThreadsWithTasks();

}

class OS1\_B extends ThreadScheduler\_B {

public void scheduleThreadsWithTasks() {

schedule();

System.*out*.println("\t\tSchedule threads with tasks in OS1.");

}

}

class OS2\_B extends ThreadScheduler\_B {

public void scheduleThreadsWithTasks() {

schedule();

System.*out*.println("\t\tSchedule threads with tasks in OS2.");

}

}

class OS3\_B extends ThreadScheduler\_B {

public void scheduleThreadsWithTasks() {

schedule();

System.*out*.println("\t\tSchedule threads with tasks in OS3.");

}

}

public class Program {

public static void main(String[] args) {

OS1\_A os1\_a = new OS1\_A();

OS2\_A os2\_a = new OS2\_A();

OS3\_A os3\_a = new OS3\_A();

System.*out*.println("[DBG] Schedule Threads with PreemptiveThreadScheduler."); os1\_a.scheduleThreadsWithTasks();

os2\_a.scheduleThreadsWithTasks();

os3\_a.scheduleThreadsWithTasks();

OS1\_B os1\_b = new OS1\_B();

OS2\_B os2\_b = new OS2\_B();

OS3\_B os3\_b = new OS3\_B();

System.*out*.println("[DBG] Schedule Threads with TimeSlicedThreadScheduler."); os1\_b.scheduleThreadsWithTasks();

os2\_b.scheduleThreadsWithTasks();

os3\_b.scheduleThreadsWithTasks();

}

}

|  |
| --- |
| The restructured version using bridge design pattern |

abstract class AbstractThreadScheduler {

public abstract void scheduleThreadsWithTasks();

}

class OS1 extends AbstractThreadScheduler {

public void scheduleThreadsWithTasks() {

System.*out*.println("\t\tSchedule threads with tasks in OS1.");

}

}

class OS2 extends AbstractThreadScheduler {

public void scheduleThreadsWithTasks() {

System.*out*.println("\t\tSchedule threads with tasks in OS2."); }

}

class OS3 extends AbstractThreadScheduler {

public void scheduleThreadsWithTasks() {

System.*out*.println("\t\tSchedule threads with tasks in OS3."); }

}

abstract class ThreadScheduler {

protected AbstractThreadScheduler tsImpl;

protected ThreadScheduler(AbstractThreadScheduler tsImpl) { // (1) Write your answer here

}

// (2) Write your answer here

public abstract ;

}

class ThreadScheduler\_A extends ThreadScheduler {

public ThreadScheduler\_A(AbstractThreadScheduler ts) {

super(ts);

}

public void schedule() {

System.*out*.println("\tA approach");

tsImpl.scheduleThreadsWithTasks();

}

}

class ThreadScheduler\_B extends ThreadScheduler {

public ThreadScheduler\_B(AbstractThreadScheduler ts) {

super(ts);

}

public void schedule() {

System.*out*.println("\tB approach");

tsImpl.scheduleThreadsWithTasks();

}

}

public class Program {

public static void main(String[] args) {

ThreadScheduler os1\_a = new ThreadScheduler\_A(new OS1());

ThreadScheduler os2\_a = new ThreadScheduler\_A(new OS2());

ThreadScheduler os3\_a = new ThreadScheduler\_A(new OS3());

System.*out*.println("[DBG] Schedule Threads with PreemptiveThreadScheduler."); os1\_a.schedule();

os2\_a.schedule();

os3\_a.schedule();

ThreadScheduler os1\_b = new ThreadScheduler\_B(new OS1());

ThreadScheduler os2\_b = new ThreadScheduler\_B(new OS2());

ThreadScheduler os3\_b = new ThreadScheduler\_B(new OS2());

System.*out*.println("[DBG] Schedule Threads with TimeSlicedThreadScheduler."); os1\_b.schedule();

os2\_b.schedule();

os3\_b.schedule();

}

}

**14. You develop an application using the builder design pattern. The application encapsulates building different representation of the product (e.g., XML and JSON). It consists of two builder classes, such as XMLBuilder and JSONBuilder. The XMLBuilder class constructs an object of XMLProduct, and the JSONBuilder does an object of JSONProduct. Write the output that the following program using builder design pattern produces.**

**(e.g., the output of System.out.println(“Hello World”) is Hello World.)**

**(6 points)**

|  |
| --- |
| The output of “System.out.println”  (e.g., the output of System.out.println(“Hello World”) is Hello World.) |
|  |

abstract class Product {

String data = "";

public String getData() {

return data;

}

public void setData(String data) {

this.data = data;

}

public String toString() {

return this.data;

}

public abstract void append(String data);

}

class XMLProduct extends Product {

public void append(String data) {

this.data += (" " + data + " ");

}

}

class JSONProduct extends Product {

@Override

public void append(String data) {

this.data += data;

}

}

abstract class Builder {

abstract Builder setName(String name);

abstract Builder setAddress(String address);

abstract Builder setPhone(String phone);

abstract void build();

abstract Product getProduct();

}

class XMLBuilder extends Builder {

XMLProduct product = new XMLProduct();

String name; String address; String phone;

public XMLBuilder setName(String name) {

this.name = name;

return this;

}

public XMLBuilder setAddress(String address) {

this.address = address;

return this;

}

public XMLBuilder setPhone(String phone) {

this.phone = phone;

return this;

}

public void build() {

product.append("<NAME>" + this.name + "</NAME>");

product.append("<ADDRESS>" + this.address + "</ADDRESS>"); product.append("<PHONE>" + this.phone + "</PHONE>");

product.setData("<XML>" + product.getData() + "</XML>"); }

public Product getProduct() {

return this.product;

}

}

class JSONBuilder extends Builder {

JSONProduct product = new JSONProduct();

String name;

String address;

String phone;

public JSONBuilder setName(String name) {

this.name = name;

return this;

}

public JSONBuilder setAddress(String address) {

this.address = address;

return this;

}

public JSONBuilder setPhone(String phone) {

this.phone = phone;

return this;

}

public void build() {

product.append("\"NAME\": \"" + this.name + "\", ");

product.append("\"ADDRESS\": \"" + this.address + "\", ");

product.append("\"PHONE\": \"" + this.phone + "\"");

product.setData("{JSON: {" + product.getData() + "}}");

}

public Product getProduct() {

return this.product;

}

}

class DocumentReader {

Builder builder;

public DocumentReader(Builder builder) {

this.builder = builder;

}

void build(String text) {

String[] splitedData = text.split(",");

String name = null, address = null, phone = null;

for (String data : splitedData) {

if (data.startsWith("NAME:")) {

name = data.substring(5);

} else if (data.startsWith("ADDRESS:")) {

address = data.substring(8);

} else if (data.startsWith("PHONE:")) {

phone = data.substring(6);

}

}

builder.setName(name).setAddress(address).setPhone(phone).build(); }

public Product getResult() {

return builder.getProduct();

}

}

public class Program {

public static void main(String[] args) {

System.*out*.println("[DBG] Convert a document to two different formats:"); String text = "NAME:customer1,ADDRESS:address1,PHONE:phone1";

XMLBuilder xmlBuilder = new XMLBuilder();

DocumentReader docReader = new DocumentReader(xmlBuilder);

docReader.build(text);

System.*out*.println("\tXML format: " + docReader.getResult());

JSONBuilder jsonBuilder = new JSONBuilder();

docReader = new DocumentReader(jsonBuilder);

docReader.build(text);

System.*out*.println("\tJSON format: " + docReader.getResult());

}

}