**OOP2 Problem Set 4: Inheritance, JFrames and Window Event handling** (relates to

section 4, units 10 to 12)

1. Explain what you understand by the OO term **inheritance**. Give *any* example you like to

illustrate your understanding.

In Java, inheritance allows us to utilise common characteristics from other classes. A Bicycle could have multiple variations, eg. RoadBikes, TandemBikes and MountainBikes but each type share some common attributes e.g. cadence, current gear, current speed. Each also have additional features that make them different. Inheritance allows us to inherit commonly used attribute and state from other classes.

class MountainBike **extends** Bicycle {

// new fields and methods defining

// a mountain bike would go here

}

2. Consider the following inheritance hierarchy diagram:



From this diagram identify,

(i) The **base** class(es) Vehicle

(ii) The **derived** class(es) Car, Bicycle

3. In terms of inheritance, give another name for

(i) the base class

(ii) the derived class

4. When creating a class in Java, in the absence of explicit inheritance from a superclass, the class

will implicitly (automatically) inherit from a “well-known” class instead. What is this “wellknown”

class called and to which Java package does it belong?

5. Imagine that you intend to create a class called **ScienceFictionBook** that must inherit

from a class called **Book**. Give the line of code for the ScienceFictionBook class

definition header in this case. You can assume that the class ScienceFictionBook has

**package access**.

6. Copy the files Student.java, Person.java and StudentTest.java to your X: drive. Analyse the

**Student.java** file and answer the following questions based on it.

(i) Which class is the base class here?

(ii) Which class is the derived class here?

(iii) Notice that the no-argument Student constructor does not contain any code that

refers to the Person constructor, for initializing those attributes derived from the

Person class. However, the Person attributes are initialized. How is this possible?

Would it have been possible to refer to the Person constructor from the Student

constructor directly? If so, what code would you insert to do this? What does this tell

you about the inheritance of constructors?

(iv) Notice that the multi-argument Student constructor has a long list of arguments,

some of which refer to the Student attributes, others which refer to attributes

defined in the Person class. Ignoring the driver program for now, is the order of the

argument list important here? Test it out by changing the order of the cnm and snum

arguments.

(v) Notice the reference to **super** in the Student multi-argument constructor. What

does “**super**” refer to and what is the purpose of this call here? Modify the

constructor by just moving the line of code involving the call to **super** below the

call to **setCourse().** Recompile the class. What happens? What does this tell you about calls involving super?

(vi) Explain briefly why the Student class contains no mutators or accessors that refer

to Person attributes.

(vii) The toString()method defined in the Student class **overrides** the

toString() defined in the Person class. Explain very briefly what you

understand by the term *overriding* and say why it is necessary for the Student class

to override this method here.

7. Take a quick look at the driver program **StudentTest.java** and answer the following questions

based on it:

i. Modify the driver program by adding code that will alter the **name** of the Student

object s2 to “Michael Caine” and display this and then retrieve and display the **age**

of the object.

ii. Rewrite this program (a) so that it sends its output to a message dialog (b) so that it

asks for the student details to be entered via input dialogs before creating the

second student object (c) so that it creates a default student object, then asks for the

details and uses mutators to set them

iii. Draw a VOPC diagram showing all classes needed for this application to run, and

all attributes and methods for the user-written classes that are included in the

diagram.

8 Explain very briefly what you understand by the term “*event-driven programming*”.

9. Consider the inheritance hierarchy chart for the Swing classes as given in your lecture notes.

What class is the *base* class for the JTextField class?

10. Using the **Java API documentation**, answer the following questions:

(i) To which *package* does the JButton class belong?

(ii) What is the **immediate** superclass of JButton?

(iii) What *interface* does JButton implement?

(iv) Name *any* class that derives from the JButton class.

(v) The JButton class inherits several methods from the AbstractButton class.

One of these is called getText(). What is the purpose of this particular method?

What is its **return type**?

(vi) The JButton class implements the Accessible interface. How many methods

are defined as part of this interface?

(vii) What class is at the very top of the Java inheritance hierarchy? To which package

does it belong?

(viii) How many constructors are defined as part of the JButton class?

(ix) Name *any* method defined in the JButton class that overrides a method defined in

another class it would otherwise inherit.

(x) Name any method that JButton inherits from the class Component.

(xi) JButton derives fields and methods from 5 different classes. Name these.

(xii) How can you tell that the class AbstractButton is non-instantiable?

11. Copy the file **ClosingWindow1.java** to your H: drive and answer the following questions:

(i) Why does the package javax.swing need to be imported in this program?

Comment out this line of code temporarily and recompile to determine why it must be imported.

(ii) Why does the package java.awt.event need to be imported in this program?

Comment out this line of code momentarily and recompile to determine why it must

be imported.

(iii) This application shows an example of *inheritance*. Which is the *base* class and which

is the *derived* class?

(iv) Explain the purpose of the call to super() in the constructor. Would the program

still compile and run if this line of code was omitted? Try it out. What happens?

(v) Explain what the purpose of the call to the setSize() method is. Would the

program still compile and run is this line of code was omitted? Try it out. What

happens?

(vi) Explain what the purpose of the call to the setVisible() method is. Would the

program still compile and run is this line of code was omitted? Try it out. What

happens?

(vii) Explain what the purpose of the call to the addWindowListener() method is.

Would the program still compile and run is this line of code was omitted? Try it out.

What happens?

(viii) This program defines a private inner class called WindowEventHandler that

overrides the default operation for the windowClosing() method. State

specifically what the objective of the overriding is in this case.

(ix) Use the **Java API documentation** to study the class WindowAdapter. Name *any*

4 methods it contains, and list all the interfaces it implements.

(x) Note that the WindowEventHandler class is private. Would it be possible to

make this a public inner-class instead? Try it out and recompile. What does it

mean access-wise to have the class defined private?

(xi) Why is inheriting from the WindowAdapter class so useful compared with having

the private class implement the WindowListener interface directly?

(xii) Note that the driver is contained within the confines of the ClosingWindow1 class

definition here. Would it have been possible to define the driver in a **separate** file

instead? Try it out and recompile.

(xiii) When you are running this program, it runs continuously until you choose to close

the window by clicking the mouse on the appropriate section of the title bar. When

this happens an event is triggered and an **event object** is created. Specifically, what

type of event object is created here? What creates this event object? Name the **type** of

the object in the application that actually “listens” for events of this type?

(xiv) Note that the constructor makes a call to the superclass constructor in order to place

some text into the title bar. However, this functionality could also be achieved by

using an appropriate **mutator** method found in the **Frame** class from which

JFrame derives. Using the **Java API documentation**, find this method and then use

it in the program in place of the super() call.

12. Write a subclass of JFrame called BicycleFrame which should be 400 x 200 pixels, located

with its top left corner approx at 100,200, and with the title “Bicycle Shop”. Write a ‘testing’

main to create and display such a frame. What about a ‘bicycle’ icon instead of a coffee cup?

Make a copy of BicycleFrame called BicycleFrame2 (note that you must change the name in 4 or

5 different places for this to work), then change the background colour of your BicycleFrame2

(have a look at the Color class to see the pre-set options and the different ways to construct a

colour of your own choice).

13. Write a GUI application called **MyJFrameWindow** that simply inherits from JFrame and

does the following in its **constructor**:

 places the following text into the window’s title bar: “My own JFrame

Application”.

 Sets the size of the window to 300 pixels wide by 200 pixels high

 Adds a listener to the window that will listen for Window events.

 Shows the window on screen

This class should contain its own private class called **WindowEventHandler** which inherits

from the WindowAdapter class and overrides the appropriate events to achieve the following:

 When an attempt is made to close the window, the user should be asked via a

dialog to confirm that they indeed wish to terminate the application, via

supplying a y (yes) or n (no) indication. If ‘y’ is entered then the application

should immediately terminate, if ‘n’ is entered it should stay visible. Try using a

JOptionPane.showConfirmDialog() for this bit. You will need to include the line

setDefaultCloseOperation(DO\_NOTHING\_ON\_CLOSE) in the constructor.

 When the user minimises the window by clicking the appropriate icon, a message

dialog should appear saying “Minimising the window”

 When the user restores (de-iconifies) the window, a message dialog should

appear saying “Restoring the window”.

Check out the list of methods given at the end of the problem sheet, some of which might be

useful for this application

14. Explain what you understand by the term “*anonymous inner class*”. Give any advantage of

using such classes and any disadvantage.

15. Alter the code that you have written in Q13 so that rather than using a private class called

WindowEventHandler, an **anonymous inner class** that achieves the same result is used. Call

the application MyJFrameWindow2.

16. Copy the file **SingleButton.java** to your x: drive and answer the following questions based

on it.

(i) Along with the javax.swing and java.awt.event packages, this GUI

application also imports **java.awt** package. Why does it need to import this one? If

you don’t know for sure, use the Java API documentation to find out or comment out

the import statement and recompile.

(ii) What is the purpose of the getContentPane() method?

(iii) Explain in your own words, the purpose of the following line of code:

cPane.setLayout(new FlowLayout());

(iv) Explain in your own words, the purpose of the following line of code:

button.addActionListener(new ButtonListener());

(v) Notice that, unlike the WindowEventHandler class, the ButtonListener

class does not inherit from another class. Instead it implements the

ActionListener interface and overrides its actionPerformed() method.

Why does it make sense that a class called ActionAdapter class doesn’t exist in the Java API, in the way that a WindowAdapter class does?

(vi) The ButtonListener class overrides the actionPerformed() method defined

in the interface ActionListener. Explain what you understand by the term

**overriding** and say why it is vital that the actionPerformed() method is

overridden. What would happen if the ButtonListener class did not contain

code that implemented the actionPerformed() method? Test it out by

commenting out the actionPerformed() method code.

(vii) Notice that the actionPerformed() method takes an argument of type

**ActionEvent**. To what **package** does this class belong?

17. Add two buttons called ‘Smile’ and ‘Frown’ to your BicycleFrame; when you click on a

button, it should tell you in a message dialog which one you clicked (what about a happy-face /

sad-face icon instead?)

18. Write a GUI application called **ButtonChaser** that places a JButton on a JFrame

window using the FlowLayout layout manager and changes its position to a **random location**

within the JFrame window each time it is pressed. The ButtonChaser class will subclass

JFrame and its constructor should carry out the following steps:

 places the following text into the window’s title bar: “The Button Chaser

Application”.

 Sets the size of the window to 275 pixels wide by 170 pixels high

 Locates the window somewhere close to the screen centre

 Set the default window-closing action to exit()

 Create the JButton, placing the text “Press Me” on it.

 Adds a **mnemonic** for the button associated with the letter ‘P’.

 Adds the JButton to the JFrame window.

 Adds an appropriate listener to the button that will listen for and be capable of

handling events fired by the button

 Shows the window on screen

This class should contain its own private class called **ButtonListener** which implements the

ActionListener interface and overrides its actionPerformed() method as follows:

 When the button is pressed, the program should relocate the button to some

random location in the JFrame window. In your code you must allow for the

fact that the title bar consumes around 30 pixels of the overall vertical screen

height and the borders consume about 15 pixels between them. This is cosmetic

but important to ensure the button doesn’t go “off” the JFrame window.

Note that the following predefined GUI methods are available to be used in the creation of this

application:

**JFrame(String txt)** – creates a JFrame window and places the information described by

txt into its title bar

**void setSize(int width, int height)** – sets the size of a component, giving it a

certain number of pixels in width and height.

**void setVisible(boolean b)** – shows the component on the screen if the value of b is

true, hides it otherwise

**void addWindowListener(WindowListener winlstner)** – adds the specified

window listener to “listen” for events that may be generated by this window

**void windowClosing(WindowEvent e)** – method called when a window is in the

process of being closed.

**void setLocation(int x, int y)** – places the component in the location on the

screen indicated by the x (horizontal) and y (vertical) coordinates. The default is (0,0) to indicate

the top-left of a window.

**double random()** – calculates a random value between zero (inclusive) and 1 (not inclusive).

This method is **static** and belongs to the class called **Math** in java.lang

**void addActionListener(ActionListener actlstner)** – adds the specified

action listener to “listen” for events that may be generated by this button

**Container getContentPane()** – returns a reference to the window pane’s “container” so

that other components may be added to it.

**void setLayout(LayoutManager lm)** – takes an object of a class that implements

interface LayoutManager as an argument e.g. FlowLayout, BorderLayout or GridLayout and sets

the layout style for the container concerned to this value.

**void add(Component comp)** – adds the component concerned to a particular container.

**void setMnemonic(char mnem)** – provides a keyboard alternative to pressing with the

mouse. The mnemonic is the argument mnem.