1. (12 marks) Consider the following UML class-relationship diagram:

In this diagram, the diamonds of the aggregation and composition relations have been
replaced by circles, and the name of each relationship is indicated by an italic capital
letter.

For each of the following statements, state whether it is true (T), false (F), or unknown
(U) relative to the diagram above.

a) (2 marks) Each Y is a component of a V. (T)

b) (2 marks) X is derived by inheritance from V. (F)

c) (2 marks) There may be 2 U associated with every Y. (F)

d) (2 marks) There is one W for every Y. (F)

e) (2 marks) There is exactly one W for every X. (T)

f) (2 marks) There is one Y for every V. (T)
2. (10 marks) Sketch the UML diagram corresponding to the following situation:

Each person at a university has a unique identification number \textbf{IDnum} and a method \textbf{getID} for accessing that ID number. There are three derived categories of people – namely, students, staff, and faculty. The university also offers a set of courses and maintains a set of pension plans for its faculty and staff. Each course may have up to 40 students and each student has between three and five associated courses. Each pension plan has exactly one associated staff or faculty member, and each staff or faculty member can have up to 3 associated pension plans. Each of the student, staff, and faculty categories have full-time and part-time variants, and these variants are derived by a combination of inheritance and implementation of interfaces specifying of constants and methods associated with part-time and full-time status.
3. (12 marks) Consider the following Java program:

```java
class tf_q3 {
    public static void main(String[] arg) {
        XObj x = new XObj(4), y;
        YObj z = new YObj(7);
        int a = 2, b = 9;
        x.printVal();
        z.printVal();
y = z;
y.printVal();
    }
}

class XObj {
    public int  a = 5, b = 6;
    public XObj() {}
    public XObj(int a) {
        a = a;
    }
    public void printVal() {
        int b = 6;
        System.out.println("Xver: "+a+" " +b);
    }
}

class YObj extends XObj {
    public int a = 8;
    public YObj(int a) {
        super();
        this.a = a;
    }
    public void printVal() {
        System.out.println("Yver: " +a+" " +b);
    }
}
```

Please give the output produced by executing this program.

```
Xver: 5 6
Yver: 7 6
Yver: 7 6
```
4. (10 marks)

a) (5 marks) Consider a Java method

```java
public static int funky(int a, boolean b, char c, byte d)
```

which has a defined return value when $1 \leq a \leq 10^6$, $b \in \{\text{true, false}\}$, $c \in \{'x', 'X', 'y', 'Y', 'z', 'Z'\}$, and $-1001 \leq d \leq 5$, and issues various exceptions in all other cases. Give a set of test cases for this method relative to the functional (black box) test-case selection criterion.

$$
\{0, 1, 2, 1000, 10^6 - 1, 10^6, 10^6 + 1\} \times \{T, F\} \times \{'x', 'X', 'y', 'Y', 'z', 'Z'\} \times \\
\{-1002, -1001, -1000, -100, -1, 0, 1, 4, 5, 6\}
$$

b) (5 marks) Consider the following Java method:

```java
public static int sonOfFunky(int a, char c){
    if (a > 2)
        if (c == 'c')
            a = (a * a * a);
        else
            a += 42;
    if (c == 'a')
        return(a);
    else
        return((int) c);
}
```

Give a set of test cases for this method relative to the structural (clear box) test-case selection criterion that ensures that each possible execution-path through this method is executed at least once.
5. (12 marks) Consider the following Java program:

```java
import javax.swing.*;
import java.awt.*;

class tf_GUI extends JFrame {
    private JPanel displayPanel1, displayPanel2;

    public tf_GUI() {
        super("GUI: tf_GUI");

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.getContentPane().setLayout(new BorderLayout());

        displayPanel1 = new JPanel();
        displayPanel1.setPreferredSize(new Dimension(300, 100));
        displayPanel1.setLayout(new FlowLayout());
        displayPanel1.add(new JLabel("Something for you: "));
        displayPanel1.add(new JButton("Push Me!"));

        displayPanel2 = new JPanel();
        displayPanel2.setPreferredSize(new Dimension(100, 100));
        displayPanel2.setLayout(new BorderLayout());
        displayPanel2.add(new JButton("N"), BorderLayout.NORTH);
        displayPanel2.add(new JButton("S"), BorderLayout.SOUTH);
        displayPanel2.add(new JButton("W"), BorderLayout.EAST);
        displayPanel2.add(new JButton("E"), BorderLayout.WEST);

        this.getContentPane().add(displayPanel1, BorderLayout.SOUTH);
        this.getContentPane().add(displayPanel2, BorderLayout.WEST);
        this.getContentPane().add(new JButton("Here!"), BorderLayout.CENTER);
    } // End of constructor method

    public void display(){
        this.pack();
        this.show();
    } // End of method display

    public static void main(String[] arg){
        tf_GUI G;

        G = new tf_GUI();
        G.display();
    } // End of main method

} // End of class tf_GUI
```

Please sketch the frame(s) produced by executing this program.
6. (64 marks)

All parts of this question refer to the classes described in your exam handout.

a) (12 marks) Give the appropriate comments and javadoc tags appearing in the places marked (A)–(D) in the code-skeleton below that are needed to reproduce the comments associated with class Sponsor and the methods getName, addEvent, and sponsoredEvent in the printout of the javadoc-produced HTML file given in your exam handout.

```java
import java.util.Vector;

(A)

public class Sponsor {

    String name, addr;
    Vector E;
    int cse = 0, eec = -1;

    ...

    (B)

    public String getName(){ ... }

    ...

    (C)

    public void addEvent(Event e) throws alrdy_event { ... }
```
public boolean sponsoredEvent(Event e){ ... }

...}

} // End of class Sponsor

Answer:

(A): /**
   * Implements operations and data associated with an event sponsor.
   * Note that the list of events being sponsored by a sponsor is
   * openended and hence is stored as a <tt>Vector</tt>-object.
   * *
   * @author Todd Wareham
   * @see Event
   */

(B): /**
   * Returns this sponsor’s name.
   */

(C): /**
   * Adds an event to the list of events sponsored by this
   * sponsor. Note that this list is stored as a
   * <tt>Vector</tt>-object.
   *
   * @param e Event to be added
   * @throws alrdy_event if <tt>e</tt> is already being sponsored
   * by this sponsor
   */

(D): /**
   * Determines if an event is being sponsored by this sponsor
   *
   * @param e Event
   * @return <tt>TRUE</tt> if <tt>e</tt> is already being
   * by this sponsor and <tt>FALSE</tt> otherwise.
   */
b) (26 marks) Give Java code for all fields associated with class StopWatch, as well as the methods StopWatch (the constructor), startEvent, markFinish, stopEvent, and eventTime associated with this class. All fields should be accessible only within this class, and all requested methods should be publicly available. You may assume the existence of all invoked exception-classes as well as the additional System method

\[
\text{public static double time()}
\]

that returns a double value denoting the number of microseconds at this method’s call-time since midnight, December 31, 1899.

```java
class StopWatch {
    private double startTime, finishTime;
    private double[] T;

    public StopWatch(int n) throws bad_param {
        if (n < 2)
            throw new bad_param();
        T = new double[n];
        for (int i = 0; i < n; i++)
            T[i] = -1.0;
        startTime = finishTime = -1.0;
    }

    public void startEvent(){
        startTime = System.time();
    }

    public double elapsedTime() throws not_started, alrdy_finished {
        if (startTime == -1.0)
            throw new not_started();
        if (finishTime != -1.0)
            throw new alrdy_finished();
        return(System.time() - startTime);
    }
}
```
public void markFinish(int an) {
    throws not_athlete, not_started, alrdy_finished, alrdy_marked {
        if (!(an >= 1) && (an <= T.length))
            throw new not_athlete();
        if (startTime == -1.0)
            throw new not_started();
        if (finishTime != -1.0)
            throw new alrdy_finished();
        if (T[an - 1] != -1.0)
            throw new alrdy_marked();
        T[an - 1] = System.time();
    }
}

public void stopEvent() {
    finishTime = System.time();
    for (int i = 0; i < T.length; i++)
        if (T[i] == -1.0)
            T[i] = finishTime;
}

public double finishTime(int an) throws not_athlete, not_started, not_finished {
    if (!(an >= 1) && (an <= T.length))
        throw new not_athlete();
    if (startTime == -1.0)
        throw new not_started();
    if (finishTime == -1.0)
        throw new not_finished();
    return(T[an - 1]);
}

public double eventTime() throws not_started, not_finished {
    if (startTime == -1.0)
        throw new not_started();
    if (finishTime == -1.0)
        throw new not_finished();
    return(finishTime - startTime);
}
c) (26 marks) Please give the Java code “skeletons” (class definitions with all fields and the first line of each method) for each of the classes Event, EventUntimed, EventTimed, EventGym, EventCyc, and EventSwm relative to the following restrictions:

- Classes Event, EventUntimed, and EventTimed cannot have associated object-instances.
- Classes EventGym, EventCyc, and EventSwm cannot have subclasses.
- All fields are accessible only within the classes in which they are defined.
- Method getPNumber is only accessible within class Event and its subclasses.
- All remaining methods are publicly available.
- No method can be overridden.

abstract public class Event {
    private Athlete[] A;
    private JudgePanel Jp;
    private Sponsor Sp;
    private double[] Sc;
    private String name;
    private int cra, aec;

    public Event(){ ... }

    public Event(int n, String en, JudgePanel Jp, Sponsor Sp)
    throws bad_param { ... }

    final public String getName(){ ... }

    final public JudgePanel getJudgePanel(){ ... }

    final public int numAthletes(){ ... }

    final public void register(Athlete a)
    throws athlete_exists, event_full { ... }

    final protected int getPNumber(Athlete a)
    throws not_athlete { ... }

    final public void startAthleteEnum()
    throws reg_unfinished { ... }
}
final public Athlete nextAthlete()
    throws no_enum, end_enum { ... }

final public boolean endAthleteEnum()
    throws no_enum { ... }

final public void setScore(Athlete a, double s)
    throws not_athlete { ... }

final public double getScore(Athlete a)
    throws not_athlete { ... }

final public int getPlace(Athlete a)
    throws not_athlete, event_unfinished { ... }

abstract class EventTimed extends Event {
    private double[] T;
    Stopwatch S;

    public EventTimed(){ ... }

    public EventTimed(int n, String en, JudgePanel Jp, 
                        Sponsor Sp, Stopwatch S)
        throws bad_param { ... }

    final public void setTime(Athlete a, double t)
        throws not_athlete, bad_time { ... }

    final public double getTime(Athlete a)
        throws not_athlete { ... }

}

abstract class EventUntimed extends Event {

    public EventUntimed(){ ... }

    public EventUntimed(int n, String en, JudgePanel Jp, 
                         Sponsor Sp)
        throws bad_param { ... }

}
final class EventGym extends EventUntimed {

    public EventGym(int n, String en, JudgePanel Jp,
                    Sponsor Sp)
        throws bad_param { ... }
}

final class EventCyc extends EventTimed {

    public EventCyc(int n, String en, JudgePanel Jp,
                    Sponsor Sp, Stopwatch S)
        throws bad_param { ... }
}

final class EventSwm extends EventTimed {

    public EventSwm(int n, String en, JudgePanel Jp,
                    Sponsor Sp, Stopwatch S)
        throws bad_param { ... }
}