

UNIVERSITY OF TORONTO MISSISSAUGA
OCTOBER 2016 MIDTERM EXAMINATION
CSC 207H5F

Introduction to Software Design

Sadia Sharmin

Arnold Rosenbloom

Duration — 2 hours

Aids: *None*

Student Number: _____

Last Name: _____

First Name: _____

Signature: _____

Do not turn this page until you have received the signal to start.

(In the meantime, please fill out the identification section above,
and read the instructions below carefully.)

This midterm examination consists of 6 questions on 16 pages (including this one), printed on both sides of the paper. **NOTE:** The end of the exam has an API and code for an earlier question. Feel free to rip these out to make it easier to refer to them.

When you receive the signal to start, please make sure that your copy of the examination is complete. Answer each question directly on the examination paper, in the space provided.

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

The University of Toronto Mississauga and you, as a student, share a commitment to academic integrity. You are reminded that you may be charged with an academic offence for possessing any unauthorized aids during the writing of an exam, including but not limited to any electronic devices with storage, such as cell phones, pagers, personal digital assistants (PDAs), iPods, and MP3 players. Unauthorized calculators and notes are also not permitted. Do not have any of these items in your possession in the area of your desk. Please turn the electronics off and put all unauthorized aids with your belongings at the front of the room before the examination begins. If any of these items are kept with you during the writing of your exam, you may be charged with an academic offence. A typical penalty may cause you to fail the course.

1: _____ / 3

2: _____ / 4

3: _____ / 7

4: _____ / 20

5: _____ / 3

6: _____ / 18

TOTAL: _____ / 55

Question 1. [3 MARKS]

List 3 advantages of using a version control system like git.

- ~~Easier for collaboration and keeping everyone on same page~~
- ~~Allow separate workflow with branches.~~
- Easier to maintain & see changes through commit history revert changes ; backup.
- ~~source on multiple systems.~~

Question 2. [4 MARKS]

What is an advantage of using SCRUM instead of a Waterfall methodology?

~~Allowing gradual update or continuous improvement to meet client's new requests~~

Briefly describe what happens in the following SCRUM meetings:

- Sprint planning meeting
 - ↳ all members, master, and owner meet up to discuss the ~~highest priority user story~~ & create a backlog. ↳ ~~put into~~
- The daily scrum meeting

~~between master & team,~~

~~they discuss their progress yesterday,~~
~~plan today,~~
~~and obstacles they are facing~~

Question 3. [7 MARKS]

Answer the following questions.

- (a) How many JugPuzzles are created as a result of the code below? 10 Just an empty array.
 up to 10 positions waiting
 to be filled with
 object puzzle.
- ```
JugPuzzle [] puzzles = new JugPuzzle[10];

Array [10]
```

- (b) Suppose we have Class Balloon with a String attribute 'color'.  
 What would be the output of the code below? green

```
Balloon b1 = new Balloon();

Balloon b2 = b1;

b1.color = "red";

b2.color = "green";

System.out.println(b1.color);
```

*class object.  
ref by memory.*

**A**  
**B**

- (c) Suppose we have classes A and B where B is a subclass of A (that is, B extends A).  
 Which of the following would return an error? Circle one.

- A myA = new B();
- B myB = new A();
- Both of the above.
- None of the above.

B is type of A  
 A is not type of B.

- (d) Suppose we have Class Shape. For each of the following, would it make sense to make the variable/method static? Circle Yes or No, then explain your reasoning.

- the variables 'x' and 'y' which store the x and y positions of a Shape instance - YES   
*the static will make them independent of the object. that they  
 are describing*
- the variable 'numShapes' which stores the total number of Shape instances that have been created  YES   
*number of shapes should be independent of shape object  
 created*
- the variable 'defaultColor' which stores a default colour of a Shape instance - YES  NO  
*it can be independent of objects as long as each  
 shape has its own color attribute.*
- the method 'toString' which gives a String representation of a Shape (stating its color, x, and y) - YES   
*this to string method's implementation  
 needs getting attributes from the shape, which  
 is more convenient when not independent of  
 the shape object in order to not be  
 confused with other toString from other  
 classes.*

**Question 4.** [20 MARKS]

**OO Java** Consider the classes/interfaces Animal, LandAnimal, WaterAnimal, CanFly, FlyingSquirrel as described below:

- An Animal has a name (String) and an age (int). You can ask an instance for its name and age by calling getName and getAge respectively. You can set the value for them by calling setName and setAge. It also has a method toString which returns "[name] is [age] years old" (where [name] and [age] are this instance's name and age values). *specifies implementation - . Not instance*
- A LandAnimal is an Animal that has a method walk which **prints out** "[name] is walking", and a method toString which **returns** "[name] is [age] years old, and I love to walk"
- WaterAnimal is an Animal that has a method swim which **prints out** "[name] is swimming", and a method toString which **returns** "[name] is [age] years old, and I love to swim"
- CanFly is an interface with a method fly
- A FlyingSquirrel is a LandAnimal that CanFly (when it flies, it **prints out** "[name] is flying").

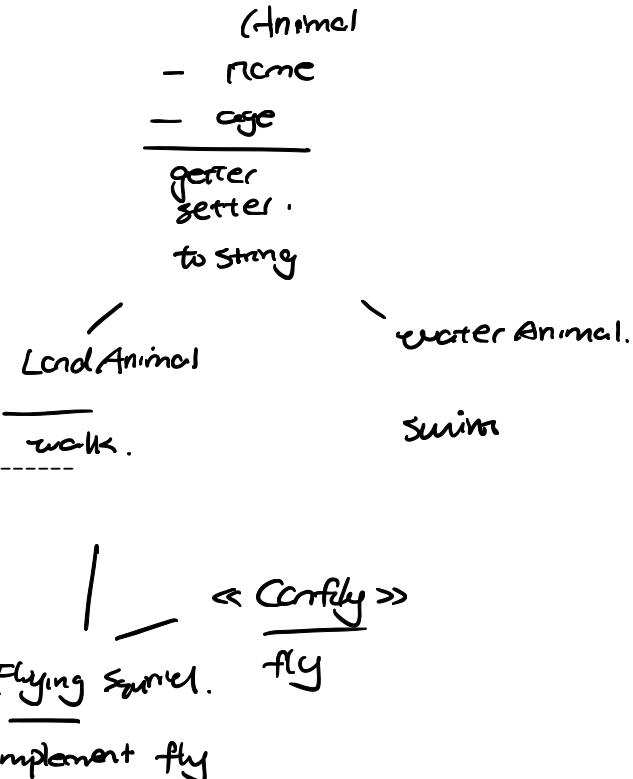
The classes above, when run with Class Zoo, results in the output at the bottom of this page.

```
public class Zoo {
 public static void main(String [] args){
 Animal a=new Animal("a", 1);
 LandAnimal l=new LandAnimal("b", 2);
 WaterAnimal w=new WaterAnimal("c", 3);
 FlyingSquirrel f=new FlyingSquirrel("d", 4);

 System.out.println(a);
 System.out.println(l);
 l.walk();
 System.out.println(w);
 w.swim();
 System.out.println(f);
 f.walk();
 f.fly();
 }
}
```

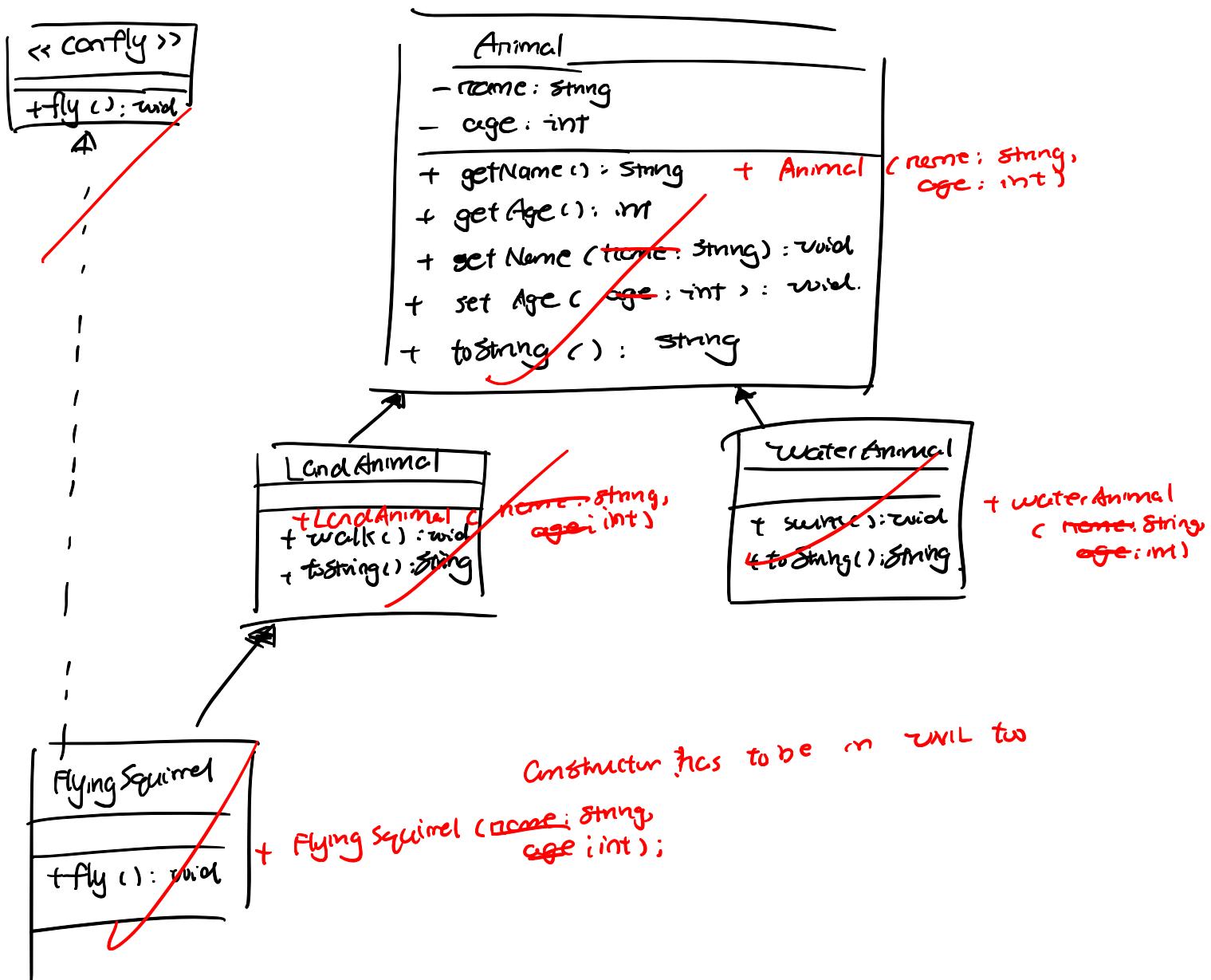
Above prints ...

```
a is 1 years old
b is 2 years old, and a land animal
b is walking
c is 3 years old, and a I love to swim
c is swimming
d is 4 years old, and a land animal
d is walking
d is flying
```



## Part (a) [5 MARKS]

Draw the UML class diagram corresponding to the description above.



## Part (b) [15 MARKS]

Write code for classes/interfaces Animal, LandAnimal, WaterAnimal, CanFly, FlyingSquirrel below.

```
public class Animal {
 private String name;
 private int age;

 public Animal (String name, int age) {
 this.name = name;
 this.age = age;
 }

 public String getName () {
 return name;
 }

 public int getAge () { return age; }

 public void setName (String name) { this.name = name; }
 public void setAge (int age) { this.age = age; }

 public String toString () {
 return (this.name + " is " + String.valueOf (getAge ()) +
 " years old");
 }
}

public interface CanFly {
 public void fly ();
}
```

(continued)

**still need constructor !!**

```
public class LandAnimal extends Animal {
 public LandAnimal (String name, int age) { super (name, age); }
 public void walk () {
 System.out.println (this + " " + get (Name) + " is walking");
 }
}
```

② override

```
public String toString () {
 return (get (Name) + " is " + string . valueOf (getAge ()))
 + " years old, and a land animal" ;
}
```

return Super . toString () + " , and a land animal "

use of parent's methods.

}

```
public class WaterAnimal extends Animal {
 public WaterAnimal (String name, int age) { super (name, age); }
 public void swim () {
 System.out.println (this + " " + get (Name) + " is swimming");
 }
}
```

③ override

```
public String toString () {
 return (get (Name) + " is " + string . valueOf (getAge ()))
 + " years old, and I (are swimming");
```

return Super . toString () + " ; and I (are swimming" )

}

}

(continued)

public class FlyingSquirrel extends LandAnimal implements CanFly {  
 public void fly () {  
 System.out.println(this.getName() + " is flying");  
 }  
}

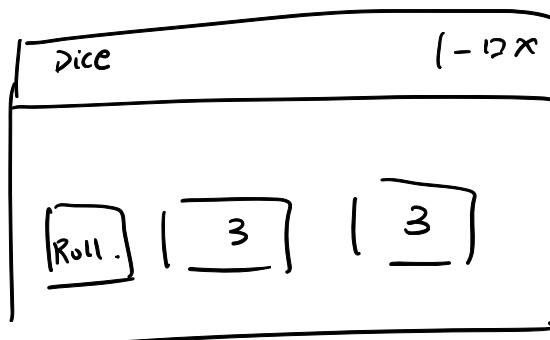
public FlyingSquirrel  
(String name,  
 int age);  
super(name, age);

**Question 5.** [3 MARKS]

GUI

**Part (a)** [3 MARKS]

The class DiceNonMVC appears in the appendix to this exam. In the space below, draw the GUI created when the main method executes. Next, describe what happens when a user interacts with the GUI.



*when clicked,  
randomly output two numbers in  
each of the two text-fields, and  
the range of random generation  
is from 1 to 6, integer*

**Question 6.** [18 MARKS]

**Model/View/Controller** Re-write DiceNonMVC using the Model/View/Controller pattern. Do this by completing/modifying the classes below. Don't worry about the imports.

**Part (a)** [3 MARKS]

Modify DiceModel as appropriate. Don't rewrite the code, just annotate it. ???

```
public class DiceModel extends Observable {
 private Random randomSource=new Random();
 private int diceValue1;
 private int diceValue2;
 public DiceModel(){
 this.roll();
 }
 public void roll(){
 this.diceValue1=randomSource.nextInt(6)+1; // 0..5->1..6
 this.diceValue2=randomSource.nextInt(6)+1; // 0..5->1..6
 }
 public int getDiceValue1(){
 return this.diceValue1;
 }
 public int getDiceValue2(){
 return this.diceValue2;
 }
}
```

*this.setChanged();  
this.notifyObservers();*

*not when constructed -  
when Action performed !!*

## Part (b) [5 MARKS]

Write class DiceController below. NOTE: See DiceApp on next page!

```

public class DiceController extends Application {
 private DiceModel model;
 private DiceView view;

 public DiceController (DiceModel model) {
 this.model = model;
 }

 public void attachView (DiceView view) {
 this.view = view;
 this.model.attach (view);
 }

 private JButton roll = new JButton ("Roll");
 private JTextField tfDie1 = new JTextField ("3");
 private JTextField tfDie2 = new JTextField ("3");

 public void run () {
 JFrame frame = new JFrame ("Dice")
 frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
 frame.getContentPane ().setLayout (new Layout ());
 JButton roll = new JButton ("Roll");
 frame.getContentPane ().add (roll);
 frame.getContentPane ().add (tfDie1);
 frame.getContentPane ().add (tfDie2);
 roll.addActionListener (this);
 frame.pack ();
 frame.setVisible (true);
 }

 @Override
 public void actionPerformed (ActionEvent e) {
 model.roll ();
 tfDie1.setText (view.getDie1Value ());
 tfDie2.setText (view.getDie2Value ());
 }
}

```

Copied.

--

## Part (c) [5 MARKS]

Write class DiceView below.

```

public class DiceView implements Observer {
 private int die1Value;
 private int die2Value;
 public DiceView () {
 this.die1Value = "5";
 this.die2Value = "3";
 }
 @Override
 public void update(Observable o, Object arg) {
 DiceModel model = (DiceModel) o;
 this.die1Value = model.getDieValue1();
 this.die2Value = model.getDieValue2();
 }
 public int getDie1Value() {
 return die1Value;
 }
 public int getDie2Value() {
 return die2Value;
 }
}

```

tfDie1 tfDie2 (Answer integrates  
\_ textfield into view.  
∴ directly update textfield.

**Part (d) [5 MARKS]**

Complete class DiceApp below.

```
public class DiceApp {
 public static void main(String [] args){
 javax.swing.SwingUtilities.invokeLater(new Runnable() {
 public void run() {
 // Create the model, view and controller, and hook them up
 DiceModel model= new DiceModel();
```

DiceController controller = new DiceController(model);

DiceView view= new DiceView();  
controller.attachView(view)  
controller.run();

```
}
});
}
```

**Short Java APIs:**

```

class Throwable:
 // the superclass of all errors and exceptions
 Throwable getCause() // the throwable that caused this throwable to get thrown
 String getMessage() // the detail message of this throwable
 StackTraceElement[] getStackTrace() // the stack trace info
class Exception extends Throwable:
 Exception(String m) // a new Exception with detail message m
 Exception(String m, Throwable c) // a new Exception with detail message m caused by c
class RuntimeException extends Exception:
 // The superclass of exceptions that don't have to be declared to be thrown
class Error extends Throwable
 // something really bad
class Object:
 String toString() // return a String representation.
 boolean equals(Object o) // = "this is o".
interface Comparable:
 // < 0 if this < o, = 0 if this is o, > 0 if this > o.
 int compareTo(Object o)
interface Iterable<T>:
 // Allows an object to be the target of the "foreach" statement.
 Iterator<T> iterator()
interface Iterator<T>:
 // An iterator over a collection.
 hasNext() // return true iff the iteration has more elements.
 next() // return the next element in the iteration.
 remove() // removes from the underlying collection the last element returned. (optional)
class Arrays:
 static sort(T[] list) // Sort list; T can be int, double, char, or Comparable
class Collections:
 static max(Collection coll) // the maximum item in coll
 static min(Collection coll) // the minimum item in coll
 static sort(List list) // sort list
interface Collection extends Iterable:
 add(E e) // add e to the Collection
 clear() // remove all the items in this Collection
 contains(Object o) // return true iff this Collection contains o
 isEmpty() // return true iff this Set is empty
 iterator() // return an Iterator of the items in this Collection
 remove(E e) // remove e from this Collection
 removeAll(Collection<?> c) // remove items from this Collection that are also in c
 retainAll(Collection<?> c) // retain only items that are in this Collection and in c
 size() // return the number of items in this Collection
 Object[] toArray() // return an array containing all of the elements in this collection
interface Set extends Collection implements Iteratable:
 // A Collection that models a mathematical set; duplicates are ignored.
class HashSet implements Set
interface List extends Collection, Iteratable:
 // A Collection that allows duplicate items.
 add(int i, E elem) // insert elem at index i
 get(int i) // return the item at index i
 remove(int i) // remove the item at index i
class ArrayList implements List
class Integer:
 static int parseInt(String s) // Return the int contained in s;
 // throw a NumberFormatException if that isn't possible
 Integer(int v) // wrap v.
 Integer(String s) // wrap s.

```

```

int intValue() // = the int value.

interface Map:
 // An object that maps keys to values.
 containsKey(Object k) // return true iff this Map has k as a key
 containsValue(Object v) // return true iff this Map has v as a value
 get(Object k) // return the value associated with k, or null if k is not a key
 isEmpty() // return true iff this Map is empty
 Set keySet() // return the set of keys
 put(Object k, Object v) // add the mapping k -> v
 remove(Object k) // remove the key/value pair for key k
 size() // return the number of key/value pairs in this Map
 Collection values() // return the Collection of values

class HashMap implement Map

class String:
 char charAt(int i) // = the char at index i.
 compareTo(Object o) // < 0 if this < o, = 0 if this == o, > 0 otherwise.
 compareToIgnoreCase(String s) // Same as compareTo, but ignoring case.
 endsWith(String s) // = "this String ends with s"
 startsWith(String s) // = "this String begins with s"
 equals(String s) // = "this String contains the same chars as s"
 indexOf(String s) // = the index of s in this String, or -1 if s is not a substring.
 indexOf(char c) // = the index of c in this String, or -1 if c does not occur.
 substring(int b) // = s[b ..]
 substring(int b, int e) // = s[b .. e)
 toLowerCase() // = a lowercase version of this String
 toUpperCase() // = an uppercase version of this String
 trim() // = this String, with whitespace removed from the ends.

class System:
 static PrintStream out // standard output stream
 static PrintStream err // error output stream
 static InputStream in // standard input stream

class PrintStream:
 print(Object o) // print o without a newline
 println(Object o) // print o followed by a newline

class Observable:
 void addObserver(Observer o) // Add o to the set of observers if it isn't already there
 void clearChanged() // Indicate that this object has no longer changed
 boolean hasChanged() // Return true iff this object has changed.
 void notifyObservers(Object arg) // If this object has changed, as indicated by
 the hasChanged method, then notify all of its observers by calling update(arg)
 and then call the clearChanged method to indicate that this object has no longer changed.
 void setChanged() // Mark this object as having been changed

interface Observer:
 void update(Observable o, Object arg) // Called by Observable's notifyObservers;
 o is the Observable and arg is any information that o wants to pass along

```

**Regular expressions:**

Here are some predefined character classes:

|    |                                                       |
|----|-------------------------------------------------------|
| .  | Any character                                         |
| \d | A digit: [0-9]                                        |
| \D | A non-digit: [^0-9]                                   |
| \s | A whitespace character: [\t\n\x0B\f\r]                |
| \S | A non-whitespace character: [^\s]                     |
| \w | A word character: [a-zA-Z_0-9]                        |
| \W | A non-word character: [^\w]                           |
| \b | A word boundary: any change from \w to \W or \W to \w |

Here are some quantifiers:

| Quantifier | Meaning                              |
|------------|--------------------------------------|
| X?         | X, once or not at all                |
| X*         | X, zero or more times                |
| X+         | X, one or more times                 |
| X{n}       | X, exactly n times                   |
| X{n,}      | X, at least n times                  |
| X{n,m}     | X, at least n; not more than m times |

## Appendix ...

```
package ca.utoronto.utm.dice;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class DiceNonMVC implements ActionListener {
 public static void main(String[] args) {
 javax.swing.SwingUtilities.invokeLater(new Runnable() {
 public void run() {
 new DiceNonMVC();
 }
 });
 }

 private Random randomSource=new Random();
 private JButton roll = new JButton("roll");
 private JTextField tfDie1 = new JTextField(3);
 private JTextField tfDie2 = new JTextField(3);
 public DiceNonMVC() {
 JFrame frame = new JFrame("Dice");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.getContentPane().setLayout(new FlowLayout());
 JButton roll = new JButton("roll");
 frame.getContentPane().add(roll);
 frame.getContentPane().add(tfDie1);
 frame.getContentPane().add(tfDie2);
 roll.addActionListener(this);
 frame.pack();
 frame.setVisible(true);
 }
 @Override
 public void actionPerformed(ActionEvent e) {
 int die1=randomSource.nextInt(6)+1; // 0..5->1..6
 int die2=randomSource.nextInt(6)+1; // 0..5->1..6
 tfDie1.setText(""+die1);
 tfDie2.setText(""+die2);
 }
}
```

Total Marks = 55