

**Model solutions to**  
**Written exam in**  
**Introductory Programming**

IT-C, January 17, 2001

## Question 1

### Question 1.1

The program prints the following seven lines, showing the 'five' and 'six' faces of a die:

```
* *  
 *  
* *  
  
* *  
* *  
* *
```

### Question 1.2

This problem can be solved in several different ways. Here is the expected solution:

```
static void show(int d) {  
    switch (d) {  
        case 1: line(0); line(2); line(0); break;  
        case 2: line(1); line(0); line(4); break;  
        case 3: line(1); line(2); line(4); break;  
        case 4: line(5); line(0); line(5); break;  
        case 5: line(5); line(2); line(5); break;  
        case 6: line(5); line(5); line(5); break;  
    }  
}
```

This is fancier:

```
static int[][] dieface = {{0,2,0},{1,0,4},{1,2,4},{5,0,5},{5,2,5},{5,5,5}};  
  
static void show(int d) {  
    for (int i=0; i<3; i=i+1)  
        line(dieface[d-1][i]);  
}
```

### Question 1.3

For a convenient solution to this problem (and the next one), one might invent an auxiliary method `rep` such that the call `rep(c, n)` prints `n` copies of the character `c`:

```
static void rep(char c, int n) {  
    for (int j=0; j<n; j=j+1)  
        System.out.print(c);  
}
```

Then the solution to the problem reads:

```
static void box(int n) {
    rep('+', n); System.out.println();
    for (int i=2; i<n; i=i+1) {
        rep('+', 1); rep(' ', n-2); rep('+', 1); System.out.println();
    }
    rep('+', n); System.out.println();
}
```

Alternatively, one may answer question 1.4 first and then answer question 1.3 by a one-liner:

```
static void box(int n)
{ fill(' ', n); }
```

## Question 1.4

Using the rep method above, the solution reads:

```
static void fill(char c, int n) {
    rep('+', n); System.out.println();
    for (int i=2; i<n; i=i+1) {
        rep('+', 1); rep(c, n-2); rep('+', 1); System.out.println();
    }
    rep('+', n); System.out.println();
}
```

## Question 2

### Question 2.1

Compute the difference between the number of days until the end of this month, and the number of days until the end of the preceding month:

```
static int monthlen(int m)
{ return days[m] - days[m-1]; }
```

### Question 2.2

This problem can be solved using `if-else`, or better, using `switch` or an array. Here is the array solution:

```
static String[] dayname = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" };

static String dayname(int wday)
{ return dayname[wday]; }
```

### Question 2.3

Take the number of days until the end of the month preceding `m`, add the date (within month `m`), and subtract 1 (to make January 1 have day number 0):

```
static int dayno(int m, int d)
{ return days[m-1] + d - 1; }
```

### Question 2.4

Since January 1, 2001 is a Monday (weekday number 0) and has day number 0 in the year, and since every week has 7 days, the weekday number can be computed as the remainder of the daynumber when dividing by 7:

```
static int weekday(int m, int d)
{ return dayno(m, d) % 7; }
```

### Question 2.5

For every date 1, 2, ..., of the month, print the name of the weekday and then the date:

```
static void prmonth(int m) {
    int len = monthlen(m);
    for (int d=1; d<=len; d++)
        System.out.println(dayname(weekday(m, d)) + " " + d);
}
```

## Question 3

### Question 3.1

Method `getCourseCount` always returns 1, and `getCourse` ignores its argument and always returns the (unique) course.

```
class SingleCourseStudent extends Student {
    Course course;

    SingleCourseStudent(String name, Course course) {
        this.name = name;
        this.course = course;
    }

    int getCourseCount()
    { return 1; }

    Course getCourse(int i)
    { return course; }
}
```

### Question 3.2

Method `getCourseCount` returns the length of the course array, and method `getCourse` returns the Course object stored in the requested element of that array:

```
class ITCStudent extends Student {
    Course[] courses;
    int studentId;

    ITCStudent(String name, Course[] courses, int studentId) {
        this.name = name;
        this.courses = courses;
        this.studentId = studentId;
    }

    int getCourseCount()
    { return courses.length; }

    Course getCourse(int i)
    { return courses[i]; }
}
```

### Question 3.3

Write implementations of the method `print` in class `SingleCourseStudent` as well as `ITCStudent`. The implementations must be written so that the main method shown above prints this when executed:

The `print` method of class `SingleCourseStudent`:

```
void print() {
    System.out.println(name + ", single-course student:");
    System.out.println(course);
    System.out.println();
}
```

The `print` method of class `ITCStudent`:

```
void print() {
    System.out.println(name + ", id " + studentId + ":");
    for (int i=0; i<courses.length; i++)
        System.out.println(courses[i]);
    System.out.println();
}
```

**Question 3.4**

Method `dayClash` should use the methods `getCourseCount` and `getCourse` to access the given student's courses. (It would be possible to implement it by using `instanceof` and by accessing the `course` and `courses` fields of the `Student` subclasses, but that leads to unmaintable code and is poor programming style).

The problem can be solved in several ways.

Solution 1: If there is a pair  $(i, j)$  of distinct course numbers ( $i < j$ ) with the same weekday, then there is a clash:

```
static boolean dayClash(Student s) {
    int count = s.getCourseCount();
    boolean clash = false;
    for (int i=0; i<count; i++)
        for (int j=i+1; j<count; j++)
            if (s.getCourse(i).getDay() == s.getCourse(j).getDay())
                clash = true;
    return clash;
}
```

Solution 2 (faster, if the number of courses is large): Create a boolean array with one element for each weekday (0–6), and note what weekdays are used. If we find that a weekday is already in use, then there is a clash:

```
static boolean dayClash2(Student s) {
    boolean[] used = new boolean[7];           // all initialized to false
    int count = s.getCourseCount();
    boolean clash = false;
    for (int i=0; i<count; i++) {
        int weekday = s.getCourse(i).getDay();
        if (used[weekday])
            clash = true;
        else
            used[weekday] = true;
    }
    return clash;
}
```

In both solutions, one can drop the variable `clash`, and instead return `true` immediately a clash is found, and return `false` after the (outer) `for`-loop.

## Question 4

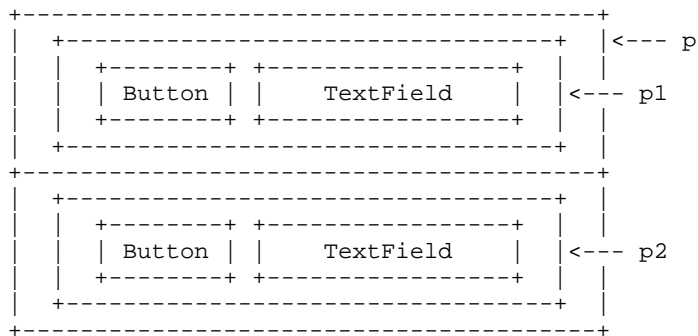
### Question 4.1

At program point `/* b */` we insert code to give `p` `BorderLayout`, to create two new Panels `p1` and `p2`, add them to `p`, and give them `BorderLayout`. Then we add a `Button` and a `TextField` to each of `p1` and `p2`:

```
p.setLayout(new BorderLayout());
Panel p1 = new Panel();
p1.setLayout(new BorderLayout());
Panel p2 = new Panel();
p2.setLayout(new BorderLayout());
p.add(p1, "North");
p.add(p2, "South");
butBush = new Button("Bush");
txtBush = new TextField(10);
txtBush.setEditable(false);
p1.add(butBush, "West"); p1.add(txtBush, "Center");
butGore = new Button("Gore");
txtGore = new TextField(10);
txtGore.setEditable(false);
p2.add(butGore, "West"); p2.add(txtGore, "Center");
add(p);
```

Alternatively, one might give `p` `GridLayout` and add the `Buttons` and `TextFields` to `p`.

The above program implements this design:



### Question 4.2

A listener increments the relevant count, and the sets the relevant textfield. This is the Bush solution (Gore is similar):

```
class BushListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        cntBush = cntBush + 1;
        txtBush.setText(Integer.toString(cntBush));
    }
}
```

This code (in `init`) creates listeners and attaches them to the buttons:

```
butBush.addActionListener(new BushListener());
butGore.addActionListener(new GoreListener());
```

## Question 4.3

Add this declaration of the field `status` to the applet (at `/* a */`):

```
TextField status;
```

Add this at the end of the `init` method:

```
status = new TextField(30);
status.setEditable(false);
add(status);
```

Declare a method `setStatus` (somewhere in `Election`) which to update the `status` text field. This method may as well update the other `TextFields`:

```
void setStatus() {
    txtBush.setText(Integer.toString(cntBush));
    txtGore.setText(Integer.toString(cntGore));
    if (cntBush > cntGore)
        status.setText("Bush is ahead by " + cntBush-cntGore);
    else if (cntBush < cntGore)
        status.setText("Gore is ahead by " + cntGore-cntBush);
    else
        status.setText("Bush and Gore are equal");
}
```

Finally, add calls to `setStatus` in the listeners. For Bush (and similarly for Gore):

```
class BushListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        cntBush = cntBush + 1;
        setStatus();
    }
}
```

## Question 4.4

Declare a new method `votes` which takes as argument the vote count difference `n`, and returns a string with 'vote' or 'votes' concatenated, as appropriate:

```
String votes(int n) {
    if (n == 1)
        return n + " vote";
    else
        return n + " votes";
}
```

Then add calls to `votes` in `setStatus`:

```
void setStatus() {
    txtBush.setText(Integer.toString(cntBush));
    txtGore.setText(Integer.toString(cntGore));
    if (cntBush > cntGore)
        status.setText("Bush is ahead by " + votes(cntBush-cntGore));
    else if (cntBush < cntGore)
        status.setText("Gore is ahead by " + votes(cntGore-cntBush));
    else
        status.setText("Bush and Gore are equal");
}
```

**Question 4.5**

The histogram class should be declared as an inner class in Election (for instance, after `/* d */`). We scale the bars so the longest one has length `size.width`, and let both bars have height `size.height/2-1` to make room for a two pixel space to separate the bars:

```
class VoteCanvas extends Canvas {
    public void paint(Graphics gr) {
        int maxcnt = Math.max(cntBush, cntGore);
        if (maxcnt > 0) {
            Dimension size = this.getSize();
            double scale = (double)size.width / maxcnt;
            gr.fillRect(0, 0, (int)(cntBush*scale), size.height/2-1);
            gr.fillRect(0, size.height/2+2, (int)(cntGore*scale), size.height/2-1);
        }
    }

    public Dimension getPreferredSize() { return new Dimension(250, 50); }
    public Dimension getMinimumSize() { return getPreferredSize(); }
}
```

At `/* a */`, a `VoteCanvas` object is created and bound to a new field `vc`:

```
Canvas vc = new VoteCanvas();
```

At `/* b */`, at the end of `init`, the `VoteCanvas` is added to the applet:

```
add(vc);
```

At the end of `setStatus`, we make sure the `VoteCanvas` gets repainted to reflect the new status:

```
vc.repaint();
```