



# **PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY**

**COURSE CODE CCE-121**

---

## **SUBMITTED TO:**

**Prof. Dr. Md Samsuzzaman**

**Department of Computer and Communication  
Engineering**

**Faculty of Computer Science and Engineering**

---

## **SUBMITTED BY:**

**Md. Sharafat Karim**

**ID: 2102024,**

**Registration No: 10151**

**Faculty of Computer Science and Engineering**

---

**Date of submission: 15 December, 2023**

**Assignment: Assignment 09**

**Assignment title: Chapter 08**

**(Deitel Java book)**

## 8.1 Fill in the blanks in each of the following statements:

- a) A(n) static **import on demand** imports all static members of a class.
- b) *String* class static method **format** is similar to method *System.out.printf*, but returns a formatted String rather than displaying a String in a command window.
- c) If a method contains a local variable with the same name as one of its class's fields, the local variable **shadows** the field in that method's scope.
- d) The public methods of a class are also known as the class's **public services** or **public interface**.
- e) A(n) **single-type-import** declaration specifies one class to import.
- f) If a class declares constructors, the compiler will not create a(n) **default constructor**.
- g) An object's **toString** method is called implicitly when an object appears in code where a String is needed.
- h) Get methods are commonly called **accessor methods**, or **query methods**.
- i) A(n) **predicate** method tests whether a condition is true or false.
- j) For every enum, the compiler generates a static method called **values** that returns an array of the enum's constants in the order in which they were declared.
- k) Composition is sometimes referred to as a(n) **has-a** relationship.
- l) A(n) **enum** declaration contains a comma-separated list of constants.
- m) A(n) **static** variable represents classwide information that's shared by all the objects of the class.
- n) A(n) **single static import** declaration imports one static member.
- o) The **principle of least privilege** states that code should be granted only the amount of privilege and access that it needs to accomplish its designated task.
- p) Keyword **final** specifies that a variable is not modifiable after initialization in a declaration or constructor.
- q) A(n) **type-import-on-demand** declaration imports only the classes that the program uses from a particular package.
- r) Set methods are commonly called **mutator methods** because they typically change a value.
- s) Use class **BigDecimal** to perform precise monetary calculations.
- t) Use the **throw** statement to indicate that a problem has occurred.

## 8.2 Explain the notion of package access in Java. Explain the negative aspects of package access.

Package access refers to the visibility of certain classes or fields, within the same package.

Using package access may limit certain features for other developers and make the code even more complicated. In certain cases, exposing certain methods can cause security issue in the entire system.

## 8.3 State an example where you can reuse the constructor of a parent class in Java.

In Java, when creating a subclass that extends a parent class, we can reuse the constructor of the parent class using the **super()** keyword.

For example, let's consider a scenario where we have a parent class called "Vehicle" with a constructor that initializes common attributes like "make" and "model." By extending this class to create a subclass "Car," we can reuse the constructor of the "Vehicle" class using "super()" to initialize shared attributes such as "make" and "model" within the "Car" subclass, streamlining the code and maintaining consistency.

## 8.4 Cylinder Class

```
1 class Cylinder {
2     float radius = 1;
3     float height = 1;
4
5     float getVolume() {
6         return (float) (Math.PI * Math.pow(radius, 2) * height);
7     }
8
9     public float getRadius() {
10        return radius;
11    }
12
13    public void setRadius(float radius) {
14        this.radius = radius;
15    }
16
17    public float getHeight() {
```

```

18     if (height < 0) {
19         throw new IllegalArgumentException("Height cannot be negative");
20     }
21     return height;
22 }
23
24 public void setHeight(float height) {
25     if (height < 0) {
26         throw new IllegalArgumentException("Height cannot be negative");
27     }
28     this.height = height;
29 }
30 }
31
32 public class Exercise_4 {
33     public static void main(String[] args) {
34         Cylinder cylinder = new Cylinder();
35         cylinder.setRadius(5);
36         cylinder.setHeight(10);
37         System.out.println("Volume of cylinder is " + cylinder.getVolume());
38     }
39 }

```

## 8.5 Modifying the Internal Data Representation of a Class

```

1  public class Time2 {
2      // private int hour; // 0 - 23
3      // private int minute; // 0 - 59
4      private int second; // 0 - 59
5      // Time2 no-argument constructor:
6      // initializes each instance variable to zero
7
8      public Time2() {
9          this(0, 0, 0); // invoke constructor with three arguments
10     }
11
12     // Time2 constructor: hour supplied, minute and second defaulted to 0

```

```
13 public Time2(int hour) {
14     this(hour, 0, 0); // invoke constructor with three arguments
15 }
16
17 // Time2 constructor: hour and minute supplied, second defaulted to 0
18 public Time2(int hour, int minute) {
19     this(hour, minute, 0); // invoke constructor with three arguments
20 }
21
22 // Time2 constructor: hour, minute and second supplied
23 public Time2(int hour, int minute, int second) {
24     if (hour < 0 || hour >= 24)
25         throw new IllegalArgumentException("hour must be 0-23");
26     if (minute < 0 || minute >= 60)
27         throw new IllegalArgumentException("minute must be 0-59");
28     if (second < 0 || second >= 60)
29         throw new IllegalArgumentException("second must be 0-59");
30     // this.hour = hour;
31     setHour(hour);
32     // this.minute = minute;
33     setMinute(minute);
34     this.second = second;
35 }
36
37 // Time2 constructor: another Time2 object supplied
38 public Time2(Time2 time) {
39     // invoke constructor with three arguments
40     this(time.getHour(), time.getMinute(), time.getSecond());
41 }
42
43 // Set Methods
44 // set a new time value using universal time;
45 // validate the data
46 public void setTime(int hour, int minute, int second) {
47     if (hour < 0 || hour >= 24)
48         throw new IllegalArgumentException("hour must be 0-23");
```

```
49     if (minute < 0 || minute >= 60)
50         throw new IllegalArgumentException("minute must be 0-59");
51     if (second < 0 || second >= 60)
52         throw new IllegalArgumentException("second must be 0-59");
53     // this.hour = hour;
54     setHour(hour);
55     // this.minute = minute;
56     setMinute(minute);
57     this.second = second;
58 }
59
60 // validate and set hour
61 public void setHour(int hour) {
62     if (hour < 0 || hour >= 24)
63         throw new IllegalArgumentException("hour must be 0-23");
64     // this.hour = hour;
65     this.second += hour * 3600;
66 }
67
68 // validate and set minute
69 public void setMinute(int minute) {
70     if (minute < 0 || minute >= 60)
71         throw new IllegalArgumentException("minute must be 0-59");
72     // this.minute = minute;
73     this.second += minute * 60;
74 }
75
76 // validate and set second
77 public void setSecond(int second) {
78     if (second < 0 || second >= 60)
79         throw new IllegalArgumentException("second must be 0-59");
80     this.second += second;
81 }
82
83 // Get Methods
84 // get hour value
```

```

85  public int getHour() {
86      return second / 3600;
87      // return hour;
88  }
89
90  // get minute value
91  public int getMinute() {
92      return (second % 3600) / 60;
93  }
94
95  // get second value
96  public int getSecond() {
97      return second % 60;
98      // return second;
99  }
100
101  // convert to String in universal-time format (HH:MM:SS)
102  public String toUniversalString() {
103      return String.format(
104          "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
105  }
106
107  // convert to String in standard-time format (H:MM:SS AM or PM)
108  public String toString() {
109      return String.format("%d:%02d:%02d %s",
110          ((getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12),
111          getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
112  }
113
114 }

```

## 8.6 Savings Account Class

```

1  class SavingsAccount {
2      private static double annualInterestRate;
3      private double savingsBalance;
4

```

```
5  public SavingsAccount(double savingsBalance) {
6      this.savingsBalance = savingsBalance;
7  }
8
9  public static void modifyInterestRate(double newInterestRate) {
10     annualInterestRate = newInterestRate;
11 }
12
13 public void calculateMonthlyInterest() {
14     savingsBalance += savingsBalance * annualInterestRate / 12;
15 }
16
17 public double getSavingsBalance() {
18     return savingsBalance;
19 }
20 }
21
22 public class Problem_6 {
23     public static void main(String[] args) {
24         SavingsAccount saver1 = new SavingsAccount(2000.00);
25         SavingsAccount saver2 = new SavingsAccount(3000.00);
26
27         SavingsAccount.modifyInterestRate(0.04);
28         for (int i = 0; i < 12; i++) {
29             saver1.calculateMonthlyInterest();
30             saver2.calculateMonthlyInterest();
31         }
32
33         System.out.printf("Saver 1 balance: %.2f\n",
34 saver1.getSavingsBalance());
35
36         System.out.printf("Saver 2 balance: %.2f\n",
37 saver2.getSavingsBalance());
38
39         SavingsAccount.modifyInterestRate(0.05);
40         saver1.calculateMonthlyInterest();
41         saver2.calculateMonthlyInterest();
42     }
43 }
```



```
39
40     System.out.printf("Saver 1 balance: %.2f\n",
saver1.getSavingsBalance());
41     System.out.printf("Saver 2 balance: %.2f\n",
saver2.getSavingsBalance());
42 }
43 }
```

## 8.7 (Enhancing Class Time2)

```
1  class Time2 {
2      private int hour; // 0 - 23
3      private int minute; // 0 - 59
4      private int second; // 0 - 59
5      // Time2 no-argument constructor:
6      // initializes each instance variable to zero
7
8      public Time2() {
9          this(0, 0, 0); // invoke constructor with three arguments
10     }
11
12     // Time2 constructor: hour supplied, minute and second defaulted to 0
13     public Time2(int hour) {
14         this(hour, 0, 0); // invoke constructor with three arguments
15     }
16
17     // Time2 constructor: hour and minute supplied, second defaulted to 0
18     public Time2(int hour, int minute) {
19         this(hour, minute, 0); // invoke constructor with three arguments
20     }
21
22     // Time2 constructor: hour, minute and second supplied
23     public Time2(int hour, int minute, int second) {
24         if (hour < 0 || hour >= 24)
25             throw new IllegalArgumentException("hour must be 0-23");
26         if (minute < 0 || minute >= 60)
27             throw new IllegalArgumentException("minute must be 0-59");
```

```
28     if (second < 0 || second >= 60)
29         throw new IllegalArgumentException("second must be 0-59");
30     this.hour = hour;
31     this.minute = minute;
32     this.second = second;
33 }
34
35 // Time2 constructor: another Time2 object supplied
36 public Time2(Time2Second time) {
37     // invoke constructor with three arguments
38     this(time.getHour(), time.getMinute(), time.getSecond());
39 }
40
41 // Set Methods
42 // set a new time value using universal time;
43 // validate the data
44 public void setTime(int hour, int minute, int second) {
45     if (hour < 0 || hour >= 24)
46         throw new IllegalArgumentException("hour must be 0-23");
47     if (minute < 0 || minute >= 60)
48         throw new IllegalArgumentException("minute must be 0-59");
49     if (second < 0 || second >= 60)
50         throw new IllegalArgumentException("second must be 0-59");
51     this.hour = hour;
52     this.minute = minute;
53     this.second = second;
54 }
55
56 // validate and set hour
57 public void setHour(int hour) {
58     if (hour < 0 || hour >= 24)
59         throw new IllegalArgumentException("hour must be 0-23");
60     this.hour = hour;
61 }
62
63 // validate and set minute
```

```
64  public void setMinute(int minute) {
65      if (minute < 0 || minute >= 60)
66          throw new IllegalArgumentException("minute must be 0-59");
67      this.minute = minute;
68  }
69
70  // validate and set second
71  public void setSecond(int second) {
72      if (second < 0 || second >= 60)
73          throw new IllegalArgumentException("second must be 0-59");
74      this.second = second;
75  }
76
77  // Get Methods
78  // get hour value
79  public int getHour() {
80      return hour;
81  }
82
83  // get minute value
84  public int getMinute() {
85      return minute;
86  }
87
88  // get second value
89  public int getSecond() {
90      return second;
91  }
92
93  // convert to String in universal-time format (HH:MM:SS)
94  public String toUniversalString() {
95      return String.format(
96          "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
97  }
98
99  // convert to String in standard-time format (H:MM:SS AM or PM)
```

```
100 public String toString() {
101     return String.format("%d:%02d:%02d %s",
102         ((getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12),
103         getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
104 }
105
106 public void incrementHour() {
107     if (hour == 23) {
108         hour = 0;
109     } else {
110         hour++;
111     }
112 }
113
114 public void incrementMinute() {
115     if (minute == 59) {
116         minute = 0;
117         incrementHour();
118     } else {
119         minute++;
120     }
121 }
122
123 public void tick() {
124     if (second == 59) {
125         second = 0;
126         incrementMinute();
127     } else {
128         second++;
129     }
130 }
131 }
132
133 public class Time {
134     public static void main(String[] args) {
135         Time2 t1 = new Time2(); //00:00:00
```

```

136     Time2 t2 = new Time2(2); // 02:00:00
137     Time2 t3 = new Time2(12, 25, 42); // 12:25:42
138
139     System.out.println(t1.toUniversalString());
140     System.out.println(t2.toUniversalString());
141     System.out.println(t3.toUniversalString());
142
143     t1.setTime(13, 27, 6);
144     t1.tick();
145     System.out.println(t1.toUniversalString());
146
147     t2.setHour(22);
148     t2.setMinute(34);
149     t2.setSecond(45);
150     t2.incrementHour();
151     System.out.println(t2.toString());
152
153     t3.setTime(23, 59, 59);
154     t3.tick();
155     System.out.println(t3.toString());
156 }
157 }

```

## 8.8 (Enhancing Class Date)

```

1 // Fig. 8.7: Date.java
2 // Date class declaration.
3 public class Date {
4     private int month; // 1-12
5     private int day; // 1-31 based on month
6     private int year; // any year
7     private static final int[] daysPerMonth = { 0, 31, 28, 31, 30, 31, 30, 31, 31,
8 30, 31, 30, 31 };
9
10    // constructor: confirm proper value for month and day given the year
11    public Date(int month, int day, int year) {
12        if (year <= 0)
13            throw new IllegalArgumentException("year (" + year + ") must be
greater than 0");

```

```

13     // check if month in range
14     if (month <= 0 || month > 12)
15         throw new IllegalArgumentException(
16             "month (" + month + ") must be 1-12");
17     // check if day in range for month
18     if (day <= 0 || (day > daysPerMonth[month] && !(month == 2 && day ==
19         29)))
20         throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
21     // check for leap year if month is 2 and day is 29
22     if (month == 2 && day == 29 && !(year % 400 == 0 || (year % 4 == 0 &&
year % 100 != 0)))
23         throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
24     this.month = month;
25     this.day = day;
26     this.year = year;
27     System.out.printf("Date object constructor for date %s%n", this);
28 }
29
30 public void nextDay() {
31     if (day == daysPerMonth[month]) {
32         day = 1;
33         if (month == 12) {
34             month = 1;
35             year++;
36         } else
37             month++;
38     } else
39         day++;
40 }
41
42 public void nextMonth() {
43     if (month == 12) {
44         month = 1;
45         year++;
46     } else
47         month++;
48 }
49
50 public void nextYear() {
51     year++;

```

```

51 }
52
53 // return a String of the form month/day/year
54 public String toString() {
55     return String.format("%d/%d/%d", month, day, year);
56 }
57
58 public static void main(String[] args) {
59     Date date = new Date(12, 30, 2020);
60     System.out.println(date);
61     date.nextMonth();
62     System.out.println(date);
63     date.nextYear();
64     System.out.println(date);
65 }
66 }
67 // end class Date

```

**8.9 Write code that generates n random numbers in the range 10 – 100. [Note: Only import the Scanner and SecureRandom classes.]**

```

1 import java.util.Scanner;
2 import java.security.SecureRandom;
3
4 public class RandGen {
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7         System.out.print("Enter the number of random numbers to generate:
8 ");
9         int num = scanner.nextInt();
10        scanner.close();
11
12        for (int i = 0; i < num; i++) {
13            System.out.printf("%d ", getRandomInt(10, 100));
14        }
15
16        private static int getRandomInt(int i, int j) {
17            return i + new SecureRandom().nextInt(j - i + 1);
18        }
19 }

```

**8.10** Write an enum type Food, whose constants (APPLE , BANANA, CARROT) take two parameters —the type (vegetable or fruit), and number of calories. Write a program to test the Food enum so that it displays the enum names and their information.

```
1 enum Food {
2     APPLE("fruit", 95), BANANA("fruit", 105), CARROT("vegetable", 25);
3
4     private final String type;
5     private final int calories;
6
7     Food(String type, int calories) {
8         this.type = type;
9         this.calories = calories;
10    }
11
12    public String getType() {
13        return type;
14    }
15
16    public int getCalories() {
17        return calories;
18    }
19 }
20
21 public class Enum {
22     public static void main(String[] args) {
23         System.out.println();
24         for (Food food : Food.values()) {
25             System.out.printf("%s: %s, %d calories%n", food, food.getType(),
26                 food.getCalories());
27         }
28     }
```

### 8.11 (Complex Numbers)

```
1 public class Complex {
2     private float real;
3     private float imaginary;
4
5     public Complex(float real, float imaginary) {
6         this.real = real;
```



```

7     this.imaginary = imaginary;
8 }
9
10 public Complex() {
11     this(0, 0);
12 }
13
14 public void addNumber(float real, float imaginary) {
15     this.real += real;
16     this.imaginary += imaginary;
17 }
18
19 public void substrNumber(float real, float imaginary) {
20     this.real = real - this.real;
21     this.imaginary = imaginary - this.imaginary;
22 }
23
24 public String toString() {
25     return String.format("%f + %fi", this.real, this.imaginary);
26 }
27
28 public static void main(String[] args) {
29     Complex complex = new Complex(1, 2);
30     System.out.println(complex);
31     complex.addNumber(3, 4);
32     System.out.println(complex);
33     complex.substrNumber(5, 6);
34     System.out.println(complex);
35 }
36 }

```

## 8.12 (Date and Time Class)

```

1 // Date class declaration.
2 class Date {
3     private int month; // 1-12
4     private int day; // 1-31 based on month

```

```

5  private int year; // any year
6  private static final int[] daysPerMonth = { 0, 31, 28, 31, 30, 31, 30, 31, 31,
30, 31, 30, 31 };
7
8  // constructor: confirm proper value for month and day given the year
9  public Date(int month, int day, int year) {
10     if (year <= 0)
11         throw new IllegalArgumentException("year (" + year + ") must be
greater than 0");
12     // check if month in range
13     if (month <= 0 || month > 12)
14         throw new IllegalArgumentException(
15             "month (" + month + ") must be 1-12");
16     // check if day in range for month
17     if (day <= 0 || (day > daysPerMonth[month] && !(month == 2 && day ==
29)))
18         throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
19     // check for leap year if month is 2 and day is 29
20     if (month == 2 && day == 29 && !(year % 400 == 0 || (year % 4 == 0 &&
year % 100 != 0)))
21         throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
22     this.month = month;
23     this.day = day;
24     this.year = year;
25     // System.out.printf("Date object constructor for date %s%n", this);
26 }
27
28 public void nextDay() {
29     if (day == daysPerMonth[month]) {
30         day = 1;
31         if (month == 12) {
32             month = 1;
33             year++;
34         } else
35             month++;
36     } else
37         day++;
38 }
39
40 public void nextMonth() {

```

```

41     if (month == 12) {
42         month = 1;
43         year++;
44     } else
45         month++;
46 }
47
48 public void nextYear() {
49     year++;
50 }
51
52 // return a String of the form month/day/year
53 public String toString() {
54     return String.format("%d/%d/%d", month, day, year);
55 }
56 }
57 // end class Date
58
59 class Time {
60     private int hour; // 0 - 23
61     private int minute; // 0 - 59
62     private int second; // 0 - 59
63     // Time2 no-argument constructor:
64     // initializes each instance variable to zero
65
66     public Time() {
67         this(0, 0, 0); // invoke constructor with three arguments
68     }
69
70     // Time2 constructor: hour supplied, minute and second defaulted to 0
71     public Time(int hour) {
72         this(hour, 0, 0); // invoke constructor with three arguments
73     }
74
75     // Time2 constructor: hour and minute supplied, second defaulted to 0
76     public Time(int hour, int minute) {
77         this(hour, minute, 0); // invoke constructor with three arguments
78     }
79
80     // Time2 constructor: hour, minute and second supplied
81     public Time(int hour, int minute, int second) {
82         if (hour < 0 || hour >= 24)

```

```

83     throw new IllegalArgumentException("hour must be 0-23");
84     if (minute < 0 || minute >= 60)
85         throw new IllegalArgumentException("minute must be 0-59");
86     if (second < 0 || second >= 60)
87         throw new IllegalArgumentException("second must be 0-59");
88     this.hour = hour;
89     this.minute = minute;
90     this.second = second;
91 }
92
93 // Time2 constructor: another Time2 object supplied
94 public Time(Time2Second time) {
95     // invoke constructor with three arguments
96     this(time.getHour(), time.getMinute(), time.getSecond());
97 }
98
99 // Set Methods
100 // set a new time value using universal time;
101 // validate the data
102 public void setTime(int hour, int minute, int second) {
103     if (hour < 0 || hour >= 24)
104         throw new IllegalArgumentException("hour must be 0-23");
105     if (minute < 0 || minute >= 60)
106         throw new IllegalArgumentException("minute must be 0-59");
107     if (second < 0 || second >= 60)
108         throw new IllegalArgumentException("second must be 0-59");
109     this.hour = hour;
110     this.minute = minute;
111     this.second = second;
112 }
113
114 // validate and set hour
115 public void setHour(int hour) {
116     if (hour < 0 || hour >= 24)
117         throw new IllegalArgumentException("hour must be 0-23");
118     this.hour = hour;
119 }
120
121 // validate and set minute
122 public void setMinute(int minute) {
123     if (minute < 0 || minute >= 60)
124         throw new IllegalArgumentException("minute must be 0-59");

```

```
125     this.minute = minute;
126 }
127
128 // validate and set second
129 public void setSecond(int second) {
130     if (second < 0 || second >= 60)
131         throw new IllegalArgumentException("second must be 0-59");
132     this.second = second;
133 }
134
135 // Get Methods
136 // get hour value
137 public int getHour() {
138     return hour;
139 }
140
141 // get minute value
142 public int getMinute() {
143     return minute;
144 }
145
146 // get second value
147 public int getSecond() {
148     return second;
149 }
150
151 // convert to String in universal-time format (HH:MM:SS)
152 public String toUniversalString() {
153     return String.format(
154         "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
155 }
156
157 // convert to String in standard-time format (H:MM:SS AM or PM)
158 public String toString() {
159     return String.format("%d:%02d:%02d %s",
160         ((getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12),
161         getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
162 }
163
164 public void incrementHour() {
165     if (hour == 23) {
166         hour = 0;
```

```

167     } else {
168         hour++;
169     }
170 }
171
172 public void incrementMinute() {
173     if (minute == 59) {
174         minute = 0;
175         incrementHour();
176     } else {
177         minute++;
178     }
179 }
180
181 public void tick() {
182     if (second == 59) {
183         second = 0;
184         incrementMinute();
185     } else {
186         second++;
187     }
188 }
189 }
190
191 public class DateAndTime {
192     private Date date;
193     private Time time;
194
195     public DateAndTime(Date date, Time time) {
196         this.date = date;
197         this.time = time;
198     }
199
200     public void tick() {
201         time.tick();
202         if (time.getHour() == 0 && time.getMinute() == 0 &&
time.getSecond() == 0) {
203             date.nextDay();
204         }
205     }
206
207     public void incrementMinute() {

```

```
208     time.incrementMinute();
209     if (time.getHour() == 0 && time.getMinute() == 0) {
210         date.nextDay();
211     }
212 }
213
214 public void incrementHour() {
215     time.incrementHour();
216     if (time.getHour() == 0) {
217         date.nextDay();
218     }
219 }
220
221 public void incrementMonth() {
222     date.nextMonth();
223 }
224
225 public void incrementYear() {
226     date.nextYear();
227 }
228
229 public String toString() {
230     return String.format("%s %s", date, time);
231 }
232
233 public static void main(String[] args) {
234     Date date = new Date(1, 5, 2023);
235     Time time = new Time(23, 59, 59);
236     DateAndTime dateAndTime = new DateAndTime(date, time);
237
238     System.out.println(dateAndTime);
239
240     dateAndTime.incrementHour();
241     dateAndTime.tick();
242     System.out.println(dateAndTime);
243 }
244 }
```