Лекция 6.1

Побитови операции в Java

Основни теми

- Цифрови системи- десетична, двоична, осмична и шестнадесетична
- Преобразувания между цифровите системи
- Отрицателни двоични числа- допълнение до две
- Оператори за по битови ***`AND, OR`***, ***`right shift`***, ***`left shift,`*** допълнение и ***`exclusive OR .`***
- Дефиниция и основни приложения- ***`class BitSet`***
- Задачи

6.1 Цифрови системи- въведение

6.2 Съкратено представяне на двоични числа като осмични и шестнадесетични

6.3 Преобразуване на осмични и шестнадесетични до двоични

6.4 Преобразуване на двоични, осмични и шестнадесетични до десетични

6.5 Преобразуване на десетични до двоични, осмични и шестнадесетични

6.6 Отрицателни двоични- допълнение до 2

6.7 Побитови операции и оператори за програмна реализация

6.8 Приложения на class BitSet- Sieve of Eratosthenes

Задачи

Литература:

Java How to Program, 9th Edition, App. E, L

6.1 Цифрови системи- въведение

Цели числа в Java програма- представяне

- **127, -67 в десетична система (основа 10)**
- **Цифрите на число в десетична система са 0, 1, 2, ..., 9**

Машинно представяне на числата

- **двоична система (основа 2)**
- **Цифрите на число в двоична система са 0, 1**

Други цифрови системи- съкратен запис на двоично

- **Осмична (основа 8) , цифри 0, 1, 2, ..., 7**
- **Шестнадесетична (основа 16) ,**
  - **цифри 0, 1, 2, ..., 9,A,B,C,D,E,F**

Позиция на цифрите определя степента на основата

6.1 Цифрови системи- цифри

Binary digit	Octal digit	Decimal digit	Hexadecimal digit
0	0	0	0
1	1	1	1
	2	2	2
	3	3	3
	4	4	4
	5	5	5
	6	6	6
	7	7	7
		8	8
		9	9
			A (decimal value of 10)
			B (decimal value of 11)
			C (decimal value of 12)
			D (decimal value of 13)
			E (decimal value of 14)
			F (decimal value of 15)

6.1 Цифрови системи- въведение

Позиция на цифрите определя степента на основата

Positional values in the binary number system			
Binary digit	1	0	1
Position name	Fours	Twos	Ones
Positional value	4	2	1
Positional value as a power of the base (2)	2^2	2^1	2^0

Positional values in the octal number system			
Decimal digit	4	2	5
Position name	Sixty-fours	Eights	Ones
Positional value	64	8	1
Positional value as a power of the base (8)	8^2	8^1	8^0

6.2 Съкратено представяне на двоични числа като осмични и шестнадесетични

Приложение на осмични и шестнадесетични- съкратен запис на двоични- чрез разделяне на групи

Binary number	Octal equivalent	Hexadecimal equivalent
100011010001	4321	8D1

Осмично

100	011	010	001
4	3	2	1

шестадесетично

1000	1101	0001
8	D	1

6.3 Преобразуване на осмични и шестнадесетични до двоични

Чрез преобразуване на цифрите в осмично или шестнадесетично число в двоичното им представяне

Пример:

Осмично 653 →110 101 011

6→ 110

5→101

3→011

Шестнадесетично FAD5 → 1111 1010 1101 0101

F→ 1111

A→1010

D→1101

5→0101

6.4 Преобразуване на двоични, осмични и шестнадесетични до десетични

Умножаваме десетичния еквивалент на всяка цифра по позиционната ѝ стойност и сумираме произведенията

$110101_{(2)} \rightarrow 53$

$7614_{(8)} \rightarrow 3890$

$AD3B_{(16)} \rightarrow 44347$

6.4 Преобразуване на двоични, осмични и шестнадесетични до десетични

Converting a binary number to decimal

Postional values:	32	16	8	4	2	1
Symbol values:	1	1	0	1	0	1
Products:	1*32=32	1*16=16	0*8=0	1*4=4	0*2=0	1*1=1
Sum:	= 32 + 16 + 0 + 4 + 0s + 1 = 53					

Converting an octal number to decimal

Positional values:	512	64	8	1
Symbol values:	7	6	1	4
Products	7*512=3584	6*64=384	1*8=8	4*1=4
Sum:	= 3584 + 384 + 8 + 4 = 3980			

Converting a hexadecimal number to decimal

Postional values:	4096	256	16	1
Symbol values:	A	D	3	B
Products	A*4096=40960	D*256=3328	3*16=48	B*1=11
Sum:	= 40960 + 3328 + 48 + 11 = 44347			

6.5 Преобразуване на десетични до двоични, осмични и шестнадесетични

Пример: Нека преобразуваме 57 $_{(10)}$ до двоично

Positional values:	32	16	8	4	2	1

Positional values:	32	16	8	4	2	1
Symbol values:	1	1	1	0	0	1

Пример: Нека преобразуваме 103 $_{(10)}$ до осмично

Positional values:	64	8	1
Symbol values:	1	4	7

6.6 Представяне на цели числа в основната памет

Целите числа се представят в двоичен вид.

Ако големината (**`byte, short, int, long`**) на едно *поле* е n бита, то с това поле може да се изразят 2^n стойности. Половината от тези стойности се използват за представянето на положителните числа (**`0, 1, 2, ...`** $2^{n-1}-1$), а другата половина за отрицателните числа. Отрицателните числа се представят като най-старшият (най-левият) бит е **`1`**

6.6 Представяне на цели числа в основната памет

Прав код

– **положителните числа** се представят стандартно с `n-1` бита, като **най-левият**, `n-`тият бит, е 0;

```
000, 001, 010, 011 = 0, 1, 2, 3
```

– **отрицателните числа** се представят стандартно с `n-1` бита, като **най-левият**, `n-`тият бит, е 1; например:

```
100, 101, 110, 111 = 0, -1, -2, -3
```


6.6 Представяне на цели числа в основната памет

Проблеми:

✓Нулата има две представяния

✓Трудно се реализират хардуерно аритметичните операции (*напр., сума на положително и отрицателно число не може да се реализира чрез проста имплементация на събиране на двоични числа*).

6.6 Представяне на цели числа в основната памет

Обратен код

– **положителните числа** се представят стандартно с **n-1** бита, като най-левият, **n**-тият бит, е **0**;

```
000, 001, 010, 011 = 0, 1, 2, 3
```

- **отрицателните числа** се представят, като се *инвертират битовете на съответното положително число*; например:

```
111, 110, 101, 100 = 0, -1, -2, -3
```

Проблем: **Нулата има две представяния**

6.6 Представяне на цели числа в основната памет

Допълнителен код

– **положителните числа** се представят стандартно с n-1 бита, като най-левият, n-тият бит, е 0 (съвпада с прав код)

```
000, 001, 010, 011 = 0, 1, 2, 3
```

-**отрицателните числа** се представят, като се *инвертират битовете на съответното положително число и към полученият резултат се добави 1*;

например

```
111, 110, 101, 100 = -1, -2, -3, -4
```


6.6 Отрицателни двоични- преобразуване в допълнителен код

Да разгледаме машинното представяне на **32**-bit цели числа в прав код. Нека

```
int value = 13;
```

32-bit представяне на тази стойност е

```
00000000 00000000 00000000 00001101
```

Отрицателната стойност на value формираме на две стъпки

6. Намираме обратния код на даденото число (“**допълнение до 1-ца**”) *с оператора за допълнение* (~):

```
onesComplementOfValue = ~value;
```

(инвертират битовете на съответното положително число)

Машинното представяне след ~value се получава като побитово се разменят битовете- единицата става нула, нулата единица:

value:

```
00000000 00000000 00000000 00001101
```

~value (обратен код т.е. “**допълнение до 1-ца**”):

```
11111111 11111111 11111111 11110010
```


6.6 Отрицателни двоични- преобразуване в допълнителен код

2. Намираме **допълнителния код** на дадената положителна стойност (“допълнение до 2- ка”) *с добавяне на 1-ца към* **`~ value`**

така допълнението до 2 на **`value`** е

```
11111111 11111111 11111111 11110011
```


6.6 Отрицателни двоични- допълнение до 2

Проверка, че допълнителния код (“допълнение до 2- ка”) на ***value*** е **отрицателната стойност** на value

value + (~value + 1) = 0 т.е. **13 + (-13) = 0**

```
  00000000 00000000 00000000 00001101
+ 11111111 11111111 11111111 11110011
-----------------------------------
  00000000 00000000 00000000 00000000
```

Ползата от “допълнение до 2” – изваждането се свежда до събиране

Въпрос: Какво е бинарното изражение на сумата на “допълнение до 2” ” на единицата с “допълнение до 2” на единицата?

6.7 Побитови операции и оператори за програмна реализация

Машинно данните се представят като последователност от битове.

Всеки бит има стойност 0 или 1

В повечето случай 8 бита формират байт- необходимата памет за променлива от тип byte.

Други типове данни изискват повече битове

short	16
int	32
long	64
float	32
double	64
char	16.

Побитовите оператори могат да **работят само с целочислени изрази** **`byte, char, short, int и long`**, но не и с изрази в плаваща запетая

6.7 Побитови операции и оператори за програмна реализация

Operator	Name	Description
&	bitwise AND	The bits in the result are set to 1 if the corresponding bits in the two operands are both 1.
|	bitwise inclusive OR	The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR	The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0.
>>	signed right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand. If the first operand is negative, 1s are filled in from the left; otherwise, 0s are filled in from the left.
>>>	unsigned right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; 0s are filled in from the left.
~	bitwise complement	All 0 bits are set to 1, and all 1 bits are set to 0.

6.7 Побитови операции и оператори за програмна реализация

Побитов AND (&) оператор.

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
1	0	0
0	1	0
1	1	1

6.7 Побитови операции и оператори за програмна реализация

Побитов OR (|) оператор.

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
1	0	1
0	1	1
1	1	1

6.7 Побитови операции и оператори за програмна реализация

Побитов exclusive OR (^) оператор.

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
1	0	1
0	1	1
1	1	0

6.7 Побитови операции и оператори за програмна реализация

Left-shift operator наляво толкова пъти колкото е зададено с десния операнд (по подразбиране 1 път) , отдясно се запълва с нули

Signed right shift operator премества битове в левия си операнд надясно колкото е зададено с десния операнд (по подразбиране 1 път) , отляво се запълва с нули , ако левия операнд е отрицателен отляво се запълва с единици

Unsigned right shift operator премества битове в левия си операнд надясно колкото е зададено с десния операнд (по подразбиране 1 път) , отляво се запълва с нули

Операторът за побитово допълнение променя всеки 0 бит в 1 и обратно в резултата.

6.7 Побитови операции и оператори за програмна реализация

Побитови операции за присвояване

Bitwise assignment operators	
&=	Bitwise AND assignment operator.
|=	Bitwise inclusive OR assignment operator.
^=	Bitwise exclusive OR assignment operator.
<<=	Left-shift assignment operator.
>>=	Signed right-shift assignment operator.
>>>=	Unsigned right-shift assignment operator.

6.7 Побитови операции за присвояване приложение

Размяна на стойностите на две целочислени променливи

```
int x= 1, y = 2;
x^=y;    // x = 3
y^=x;    // y = 1
x^=y;    // x = 2
```

// или по- кратко

```
y^=x^=y;  // y = 1, x = 3
x^=y;     // x = 2, y = 1
```


6.7 Побитови операции и оператори за програмна реализация

Пример 1 :

Редове *10–12* въвеждат се цяло число от **стандартен вход**

Цялото число се извежда в неговата **двоична форма** в групи по ***8 бита***

Често побитовия *AND* оператор се използва с операнд наричан маска (*mask*) целочислена стойност, в която битове на определени позиции са зададени с 6. Маска се използва за скриване на някои битове и същевременно за избиране на други битове.

На **ред *18***, променливата за маскиране ***displayMask*** е инициализирана с

1 << 31, или ***10000000 00000000 00000000 00000000***

Редове ***21–30*** **генерират текстовото представяне** на двоичното число

Ред ***24*** използва побитов ***AND*** operator за прочитане на битовете в променливата **input** чрез ***AND*** сравнение с ***displayMask***.

За преминаване към бит на следваща позиция в ***input*** се използва ***left-shift operator .***

6.7 Побитови операции и оператори за програмна реализация

```
1  // Fig. L.2: PrintBits.java
2  // Printing an unsigned integer in bits.
3  import java.util.Scanner;
4
5  public class PrintBits
6  {
7     public static void main( String args[] )
8     {
9        // get input integer
10       Scanner scanner = new Scanner( System.in );
11       System.out.println( "Please enter an integer:" );
12       int input = scanner.nextInt();
13
14       // display bit representation of an integer
15       System.out.println( "\nThe integer in bits is:" );
16
17       // create int value with 1 in leftmost bit and 0s elsewhere
18       int displayMask = 1 << 31;
```


6.7 Побитови операции и оператори за програмна реализация

```
20          // for each bit display 0 or 1
21          for ( int bit = 1; bit <= 32; bit++ )
22          {
23             // use displayMask to isolate bit
24             System.out.print( ( input & displayMask ) == 0 ? '0' : '1' );
25
26             input <<= 1; // shift value one position to left
27
28             if ( bit % 8 == 0 )
29                System.out.print( ' ' ); // display space every 8 bits
30          } // end for
31       } // end main
32    } // end class PrintBits
```


6.7 Побитови операции и оператори за програмна реализация

```
Please enter an integer:
0

The integer in bits is:
00000000 00000000 00000000 00000000
```

```
Please enter an integer:
-1

The integer in bits is:
11111111 11111111 11111111 11111111
```

```
Please enter an integer:
65535

The integer in bits is:
00000000 00000000 11111111 11111111
```


6.7 Побитови операции и оператори за програмна реализация

Пример 2:**Тестване на различни побитови операции**

6. Избира се операция за тестване

6. **AND**
2. **Inclusive OR**
3. **Exclusive OR**
4. **Complement**
5. **Exit**

2 Въвеждат се данни от **стандартен вход и се изпълнява избрания оператор**

3. Извежда се резултата като се използва визуализиране на побитовата операция с двоично представяне на операндите

6.7 Побитови операции и оператори за програмна реализация

```
1  // Fig. L.4: MiscBitOps.java
2  // Using the bitwise operators.
3  import java.util.Scanner;
4
5  public class MiscBitOps
6  {
7     public static void main( String args[] )
8     {
9        int choice = 0; // store operation type
10       int first = 0; // store first input integer
11       int second = 0; // store second input integer
12       int result = 0; // store operation result
13       Scanner scanner = new Scanner( System.in ); // create Scanner
14
15       // continue execution until user exit
16       while ( true )
17       {
18          // get selected operation
19          System.out.println( "\n\nPlease choose the operation:" );
20          System.out.printf( "%s%s", "1--AND\n2--Inclusive OR\n",
21             "3--Exclusive OR\n4--Complement\n5--Exit\n" );
22          choice = scanner.nextInt();
```


6.7 Побитови операции и оператори за програмна реализация

```
23
24          // perform bitwise operation
25          switch ( choice )
26          {
27             case 1: // AND
28                System.out.print( "Please enter two integers:" );
29                first = scanner.nextInt(); // get first input integer
30                BitRepresentation.display( first );
31                second = scanner.nextInt(); // get second input integer
32                BitRepresentation.display( second );
33                result = first & second; // perform bitwise AND
34                System.out.printf(
35                   "\n\n%d & %d = %d", first, second, result );
36                BitRepresentation.display( result );
37                break;
38             case 2: // Inclusive OR
39                System.out.print( "Please enter two integers:" );
40                first = scanner.nextInt(); // get first input integer
41                BitRepresentation.display( first );
42                second = scanner.nextInt(); // get second input integer
43                BitRepresentation.display( second );
44                result = first | second; // perform bitwise inclusive OR
45                System.out.printf(
46                   "\n\n%d | %d = %d", first, second, result );
47                BitRepresentation.display( result );
48                break;
```


6.7 Побитови операции и оператори за програмна реализация

```
49                case 3: // Exclusive OR
50                   System.out.print( "Please enter two integers:" );
51                   first = scanner.nextInt(); // get first input integer
52                   BitRepresentation.display( first );
53                   second = scanner.nextInt(); // get second input integer
54                   BitRepresentation.display( second );
55                   result = first ^ second; // perform bitwise exclusive OR
56                   System.out.printf(
57                      "\n\n%d ^ %d = %d", first, second, result );
58                   BitRepresentation.display( result );
59                   break;
60                case 4: // Complement
61                   System.out.print( "Please enter one integer:" );
62                   first = scanner.nextInt(); // get input integer
63                   BitRepresentation.display( first );
64                   result = ~first; // perform bitwise complement on first
65                   System.out.printf( "\n\n~%d = %d", first, result );
66                   BitRepresentation.display( result );
67                   break;
```


6.7 Побитови операции и оператори за програмна реализация

```
4  public class BitRepresentation
5  {
6     // display bit representation of specified int value
7     public static void display( int value )
8     {
9        System.out.printf( "\nBit representation of %d is: \n", value );
10
11       // create int value with 1 in leftmost bit and 0s elsewhere
12       int displayMask = 1 << 31;
13
14       // for each bit display 0 or 1
15       for ( int bit = 1; bit <= 32; bit++ )
16       {
17          // use displayMask to isolate bit
18          System.out.print( ( value & displayMask ) == 0 ? '0' : '1' );
19
20          value <<= 1; // shift value one position to left
21
22          if ( bit % 8 == 0 )
23             System.out.print( ' ' ); // display space every 8 bits
24       } // end for
25    } // end method display
26 } // end class BitRepresentation
```


6.7 Побитови операции и оператори за програмна реализация

```
4  public class BitRepresentation
5  {
6     // display bit representation of specified int value
7     public static void display( int value )
8     {
9        System.out.printf( "\nBit representation of %d is: \n", value );
10
11       // create int value with 1 in leftmost bit and 0s elsewhere
12       int displayMask = 1 << 31;
13
14       // for each bit display 0 or 1
15       for ( int bit = 1; bit <= 32; bit++ )
16       {
17          // use displayMask to isolate bit
18          System.out.print( ( value & displayMask ) == 0 ? '0' : '1' );
19
20          value <<= 1; // shift value one position to left
21
22          if ( bit % 8 == 0 )
23             System.out.print( ' ' ); // display space every 8 bits
24       } // end for
25    } // end method display
26 } // end class BitRepresentation
```


6.7 Побитови операции и оператори за програмна реализация

```
Please choose the operation:
1--AND
2--Inclusive OR
3--Exclusive OR
4--Complement
5--Exit
1
Please enter two integers:65535 1

Bit representation of 65535 is:
00000000 00000000 11111111 11111111
Bit representation of 1 is:
00000000 00000000 00000000 00000001

65535 & 1 = 1
Bit representation of 1 is:
00000000 00000000 00000000 00000001
```


6.7 Побитови операции и оператори за програмна реализация

```
Please choose the operation:
1--AND
2--Inclusive OR
3--Exclusive OR
4--Complement
5--Exit
2
Please enter two integers:15 241

Bit representation of 15 is:
00000000 00000000 00000000 00001111
Bit representation of 241 is:
00000000 00000000 00000000 11110001

15 | 241 = 255
Bit representation of 255 is:
00000000 00000000 00000000 11111111
```


6.7 Побитови операции и оператори за програмна реализация

```
Please choose the operation:
1--AND
2--Inclusive OR
3--Exclusive OR
4--Complement
5--Exit
3
Please enter two integers:139 199

Bit representation of 139 is:
00000000 00000000 00000000 10001011
Bit representation of 199 is:
00000000 00000000 00000000 11000111

139 ^ 199 = 76
Bit representation of 76 is:
00000000 00000000 00000000 01001100
```


6.7 Побитови операции и оператори за програмна реализация

```
Please choose the operation:
1--AND
2--Inclusive OR
3--Exclusive OR
4--Complement
5--Exit
4
Please enter one integer:21845

Bit representation of 21845 is:
00000000 00000000 01010101 01010101

~21845 = -21846
Bit representation of -21846 is:
11111111 11111111 10101010 10101010
```


6.7 Побитови операции и оператори за програмна реализация

```
Please choose the shift operation:
1--Left Shift (<<)
2--Signed Right Shift (>>)
3--Unsigned Right Shift (>>>)
4--Exit
1
Please enter an integer to shift:
1

1 << 1 = 2
Bit representation of 1 is:
00000000 00000000 00000000 00000001
Bit representation of 2 is:
00000000 00000000 00000000 00000010
```


6.7 Побитови операции и оператори за програмна реализация

```
Please choose the shift operation:
1--Left Shift (<<)
2--Signed Right Shift (>>)
3--Unsigned Right Shift (>>>)
4--Exit
2
Please enter an integer to shift:
-2147483648

-2147483648 >> 1 = -1073741824
Bit representation of -2147483648 is:
10000000 00000000 00000000 00000000
Bit representation of -1073741824 is:
11000000 00000000 00000000 00000000
```


6.7 Побитови операции и оператори за програмна реализация

```
Please choose the shift operation:
1--Left Shift (<<)
2--Signed Right Shift (>>)
3--Unsigned Right Shift (>>>)
4--Exit
3
Please enter an integer to shift:
-2147483648

-2147483648 >>> 1 = 1073741824
Bit representation of -2147483648 is:
10000000 00000000 00000000 00000000
Bit representation of 1073741824 is:
01000000 00000000 00000000 00000000
```


6.8 Приложения на class BitSet- Sieve of Eratosthenes

class BitSet

Предназначен за създаване и обработка на *bit sets*, които основно се използват при работа с *boolean* флагове или маски

Може динамично да променя броя на битовете в множеството- битове се добавят при нужда и обектите от *BitSet* могат да нараснат, за да съхранят новите елементи

class BitSet има два конструктора

- конструктор по подразбиране, създава празен BitSet
- Конструктор за общо ползване, който с цяло число за аргумент задава броя на елементите в обекта от *BitSet*.

По подразбиране всеки елемент е *false* и представя бит със стойност *0*.

Ако един елемент трябва да е *true* (или да е "on") се извиква на метод *set* на с *index* указващ позицията на еменета в *BitSet* обекта- така представя бит със стойност 6.

Ако един елемент трябва да е *false* (или да е "off") се извиква на метод *clear* с *index* указващ позицията на елемента в *BitSet* обекта- така представя бит със стойност 0.

За намиране на текущата стойност на елемент в BitSet се използва *BitSet* метода *get*, който се извиква с *index* указващ позицията му в *BitSet* за четене и връща *boolean* стойност, определяща дали този елемент е on (*true*) или off (*false*).

6.8 Приложения на class BitSet- Sieve of Eratosthenes (*Ситото на Ератостен*)

class BitSet – (*вектор от битове, който може да расте според нуждите, всеки елемент на вектора има булева стойност*)

Методи за побитови операции- `and, or, xor`

Пример:

Нека b1 и b2 са BitSets, тогава

```
b1.and( b2 );
```

Извършва побитов логически `AND` между `BitSets b1` и `b2`. Резултатът се присвоява на `b1`- bit set `b1` се променя така че всеки от елементите има `true` само когато първоначалната стойност е била `true` и тази стойност е съвпадала със стойността на съответния елемент в `b2`

Когато `b2` има повече елементи от `b1`, допълнителните елементи на `b2` се игнорират. Така размерът на `b1` остава непроменен

Побитови логически `OR` и `exclusive OR` се изпълняват с командите

```
b1.or( b2 );
b1.xor( b2 );
```


6.8 Приложения на class BitSet- Sieve of Eratosthenes

class BitSet - методи

method ***size*** връща броя на битовете в BitSet.

method ***equals*** **сравнява** с два ***BitSets*** за равенство- два ***BitSet*** обекта са равни само, когато ако са равни побитово

method ***toString*** създава ***String*** представяне на елементите на ***BitSet*** обекта

6.8 Приложения на class BitSet- Sieve of Eratosthenes

Пример:

алгоритъм за намиране на прости числа

- Sieve of Eratosthenes

1. **Задава се интервал за претърсване на наличие на прости числа (2- 1023)**
2. **Инициализира** се BitSet обект от 1024 елемента на стойност ON
3. Започвайки с 2 , **елиминираме** всички числа, които имат множители като задаваме бита на съответната позиция да е OFF
4. **Разпечатваме** тези елементи на BitSet обекта , които са ON

6.8 Приложения на class BitSet- Sieve of Eratosthenes стъпки 1-2

```
1  // Fig. I.10: BitSetTest.java
2  // Using a BitSet to demonstrate the Sieve of Eratosthenes.
3  import java.util.BitSet;
4  import java.util.Scanner;
5
6  public class BitSetTest
7  {
8     public static void main( String args[] )
9     {
10       // get input integer
11       Scanner scanner = new Scanner( System.in );
12       System.out.println( "Please enter an integer from 2 to 1023" );
13       int input = scanner.nextInt();
14
15       // perform Sieve of Eratosthenes
16       BitSet sieve = new BitSet( 1024 );
17       int size = sieve.size();
18
19       // set all bits from 2 to 1023
20       for ( int i = 2; i < size; i++ )
21          sieve.set( i );
```


6.8 Приложения на class BitSet- Sieve of Eratosthenes стъпка 3

```
22
23          // perform Sieve of Eratosthenes
24          int finalBit = ( int ) Math.sqrt( size );
25
26          for ( int i = 2; i < finalBit; i++ )
27          {
28             if ( sieve.get( i ) )
29             {
30                for ( int j = 2 * i; j < size; j += i )
31                   sieve.clear( j );
32             } // end if
33          } // end for
34
35          int counter = 0;
36
```


6.8 Приложения на class BitSet- Sieve of Eratosthenes стъпка 4

```
37          // display prime numbers from 2 to 1023
38          for ( int i = 2; i < size; i++ )
39          {
40             if ( sieve.get( i ) )
41             {
42                System.out.print( String.valueOf( i ) );
43                System.out.print( ++counter % 7 == 0 ? "\n" : "\t" );
44             } // end if
45          } // end for
46
47          // display result
48          if ( sieve.get( input ) )
49             System.out.printf( "\n%d is a prime number", input );
50          else
51             System.out.printf( "\n%d is not a prime number", input );
52       } // end main
53    } // end class BitSetTest
```


6.8 Приложения на class BitSet- Sieve of Eratosthenes – изходни резултати

```
Please enter an integer from 2 to 1023
773
2       3       5       7       11      13      17
19      23      29      31      37      41      43
47      53      59      61      67      71      73
79      83      89      97      101     103     107
109     113     127     131     137     139     149
151     157     163     167     173     179     181
191     193     197     199     211     223     227
229     233     239     241     251     257     263
269     271     277     281     283     293     307
311     313     317     331     337     347     349
353     359     367     373     379     383     389
397     401     409     419     421     431     433
439     443     449     457     461     463     467
479     487     491     499     503     509     521
523     541     547     557     563     569     571
577     587     593     599     601     607     613
617     619     631     641     643     647     653
659     661     673     677     683     691     701
709     719     727     733     739     743     751
757     761     769     773     787     797     809
811     821     823     827     829     839     853
857     859     863     877     881     883     887
907     911     919     929     937     941     947
953     967     971     977     983     991     997
1009    1013    1019    1021
773 is a prime number
```


6.9 Представяне на многоцифрени числови константи

```
// Подчертаване е разрешено
// Константите се четат по- лесно
   int phoneNumber = 123_123_1234;
   long creditCardNumber =
                        123_123_1234_1234_1234L;
```


6.9 Ново в JDK 7

```
//Представяне в String на шестнадесетично число
String decToHexString =
        Integer.toHexString(Integer.MAX_VALUE);
// Подчертаване при шестнадесетична константа
int hexEquivalentToIntegerMax = 0x7f_ff_ff_ff;
// Представяне в String на двоично число
    String decToBinaryString =
                    Integer.toBinaryString(101);
// Подчертаване при двоична константа
    int binaryValue= 0b000_110_0101;
```


Задачи

Основни термини

Терминът за означаване на съставно съобщение, включващо полезни данни и допълнителни битове за проверка в теорията на кодирането се нарича *codeword (ключова дума)*.

Дефиниция. *Минималният брой от побитови разлики на две ключови думи е известна като разстояние по Hamming между двете ключови думи*

Например, да разгледаме схема за кодиране със седем бита за данните и един бит за сравнение на четност на отделна ключова дума. Ако сумата на битовете на данниете е четно число то бита за четност е нула и той е единица, когато сумата от битовете е нечетно число.

0000000 0

0000001 1

0000010 1

0000011 0

Може да се види, че при тази схема за кодиране разстоянието по *Hamming* е 2, тъй като всяка ключова дума се различава от останалите по две битови позиции

Задачи

Задача No. 1

Напишете Java приложение което да прочита две осем битови числа от стандартен вход и да пресмята разстоянието по Хаминг. Да се извеждат числата в двоичен вид и намереното разстояние по Хаминг.

Упътване: Използвайте bitwise shift.

Задача No. 2

Напишете програма, която да изписва битовете на въведено цяло число в обратен ред.

Да се изведе в двоичен вид зададеното число, преобразуваното число, а също и да се изведе преобразуваното число в десетичен вид.

Задачи

Задача No. 3

За по- бързо сортиране на писмата, Американската пощенска служба е въвела баркодове за означаване на ZIP кода на компаниите, които изпращат големи количества поща. Схемата за кодиране на трицифрени.ZIP кодове посредством пет цифри (*0 и 1*) е показана на следващшя слайд:

Така, всяка от трите цифри `(0, 1 ,2,...,9)` на ZIP код се представя с петте коефициента (*0 и 1*) на базовия код по горната таблица. Например, десетичната цифра 1 срещната в ZIP код се представя като

```
1= 0*7 + 0*4+ 0* 2 + 1* 1 + 1*0
```

и съответния баркод за тази цифра е `00011` (*вижте коефициентите за цифрата 1 по приложената таблица*).

Напишете JavaFX приложение, което въвежда от потребителя трицифрен ZIP код посредством `Alert` диалогов прозорец и извежда на стандартния изход съответния баркод . Използвайте символ ':' за означаване на късите линии на баркода(съответстват на нула в баркода) и символа '|' за означаване на дългите линии на баркода (съответстват на единица в баркода). Така, ZIP кода `111` ще трябва да се изобрази като

```
: : : | | : : : | |: : : | |
```


Задачи

Number/base code	7	4	2	1	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	1	0	0	0	1
8	1	0	0	1	0
9	1	0	1	0	0
0	1	1	0	0	0