JAVA - THE BITSET CLASS

A BitSet class creates a special type of array that holds bit values. The BitSet array can increase in size as needed. This makes it similar to a vector of bits.

This is a legacy class but it has been completely re-engineered in Java 2, version 1.4.

The BitSet defines two constructors. The first version creates a default object:

```
BitSet( )
```

The second version allows you to specify its initial size, i.e., the number of bits that it can hold. All bits are initialized to zero.

```
BitSet(int size)
```

BitSet implements the Cloneable interface and defines the methods listed in table below:

SN	Methods with Description
1	**void and(BitSet bitSet)** ANDs the contents of the invoking BitSet object with those specified by bitSet. The result is placed into the invoking object.
2	**void andNot(BitSet bitSet)** For each 1 bit in bitSet, the corresponding bit in the invoking BitSet is cleared.
3	**int cardinality( )** Returns the number of set bits in the invoking object.
4	**void clear( )** Zeros all bits.
5	**void clear(int index)** Zeros the bit specified by index.
6	**void clear(int startIndex, int endIndex)** Zeros the bits from startIndex to endIndex.1.
7	**Object clone( )** Duplicates the invoking BitSet object.
8	**boolean equals(Object bitSet)** Returns true if the invoking bit set is equivalent to the one passed in bitSet. Otherwise, the method returns false.
9	**void flip(int index)**

Reverses the bit specified by index. (

10 **void flip(int startIndex, int endIndex)**

Reverses the bits from startIndex to endIndex.1.

11 **boolean get(int index)**

Returns the current state of the bit at the specified index.

12 **BitSet get(int startIndex, int endIndex)**

Returns a BitSet that consists of the bits from startIndex to endIndex.1. The invoking object is

not changed.

13 **int hashCode( )**

Returns the hash code for the invoking object.

14 **boolean intersects(BitSet bitSet)**

Returns true if at least one pair of corresponding bits within the invoking object and bitSet are 1.

15 **boolean isEmpty( )**

Returns true if all bits in the invoking object are zero.

16 **int length( )**

Returns the number of bits required to hold the contents of the invoking BitSet. This value is determined by the location of the last 1 bit.

17 **int nextClearBit(int startIndex)**

Returns the index of the next cleared bit, (that is, the next zero bit), starting from the index specified by startIndex

18 **int nextSetBit(int startIndex)**

Returns the index of the next set bit (that is, the next 1 bit), starting from the index specified by startIndex. If no bit is set, .1 is returned.

19 **void or(BitSet bitSet)**

ORs the contents of the invoking BitSet object with that specified by bitSet. The result is placed into the invoking object.

20 **void set(int index)**

Sets the bit specified by index.

21 **void set(int index, boolean v)**

Sets the bit specified by index to the value passed in v. true sets the bit, false clears the bit.

22 **void set(int startIndex, int endIndex)**

Sets the bits from startIndex to endIndex.1.

23 **void set(int startIndex, int endIndex, boolean v)**

Sets the bits from startIndex to endIndex.1, to the value passed in v. true sets the bits, false clears the bits.

24 **int size( )**

Returns the number of bits in the invoking BitSet object.

25 **String toString( )**

Returns the string equivalent of the invoking BitSet object.

26 **void xor(BitSet bitSet)**

XORs the contents of the invoking BitSet object with that specified by bitSet. The result is placed into the invoking object

Example:

The following program illustrates several of the methods supported by this data structure:

```
import java.util.BitSet;

public class BitSetDemo {

  public static void main(String args[]) {
     BitSet bits1 = new BitSet(16);
     BitSet bits2 = new BitSet(16);

     // set some bits
     for(int i=0; i<16; i++) {
        if((i%2) == 0) bits1.set(i);
        if((i%5) != 0) bits2.set(i);
     }
     System.out.println("Initial pattern in bits1: ");
     System.out.println(bits1);
     System.out.println("\nInitial pattern in bits2: ");
     System.out.println(bits2);

     // AND bits
     bits2.and(bits1);
     System.out.println("\nbits2 AND bits1: ");
     System.out.println(bits2);

     // OR bits
     bits2.or(bits1);
     System.out.println("\nbits2 OR bits1: ");
     System.out.println(bits2);

     // XOR bits
     bits2.xor(bits1);
     System.out.println("\nbits2 XOR bits1: ");
     System.out.println(bits2);
  }
}
```

This would produce the following result:

```
Initial pattern in bits1:
{0, 2, 4, 6, 8, 10, 12, 14}

Initial pattern in bits2:
{1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14}

bits2 AND bits1:
{2, 4, 6, 8, 12, 14}

bits2 OR bits1:
{0, 2, 4, 6, 8, 10, 12, 14}

bits2 XOR bits1:
{}
```