Java SE Documentation

Binary Literals

In Java SE 7, the integral types (`byte`, `short`, `int`, and `long`) can also be expressed using the binary number system. To specify a binary literal, add the prefix `0b` or `0B` to the number. The following examples show binary literals:

```
// An 8-bit 'byte' value:
byte aByte = (byte)0b00100001;

// A 16-bit 'short' value:
short aShort = (short)0b1010000101000101;

// Some 32-bit 'int' values:
int anInt1 = 0b10100001010001011010000101000101;
int anInt2 = 0b101;
int anInt3 = 0B101; // The B can be upper or lower case.

// A 64-bit 'long' value. Note the "L" suffix:
long aLong = 0b1010000101000101101000010100010110100001010001011010000101000101L;
```

Binary literals can make relationships among data more apparent than they would be in hexadecimal or octal. For example, each successive number in the following array is rotated by one bit:

```
public static final int[] phases = {
  0b00110001,
  0b01100010,
  0b11000100,
  0b10001001,
  0b00010011,
  0b00100110,
  0b01001100,
  0b10011000
}
```

In hexadecimal, the relationship among the numbers is not readily apparent:

```
public static final int[] phases = {
    0x31, 0x62, 0xC4, 0x89, 0x13, 0x26, 0x4C, 0x98
}
```

You can use binary integral constants in code that you can verify against a specifications document, such as a simulator for a hypothetical 8-bit microprocessor:

```
public State decodeInstruction(int instruction, State state) {
  if ((instruction & 0b11100000) == 0b00000000) {
    final int register = instruction & 0b00001111;
    switch (instruction & 0b11110000) {
      case 0b00000000: return state.nop();
      case 0b00010000: return state.copyAccumTo(register);
      case 0b00100000: return state.addToAccum(register);
      case 0b00110000: return state.subFromAccum(register);
      case 0b01000000: return state.multiplyAccumBy(register);
      case 0b01010000: return state.divideAccumBy(register);
      case 0b01100000: return state.setAccumFrom(register);
      case 0b01110000: return state.returnFromCall();
      default: throw new IllegalArgumentException();
    }
  } else {
    final int address = instruction & 0b00011111;
    switch (instruction & 0b11100000) {
      case 0b00100000: return state.jumpTo(address);
      case 0b01000000: return state.jumpIfAccumZeroTo(address);
      case 0b01000000: return state.jumpIfAccumNonzeroTo(address);
      case 0b01100000: return state.setAccumFromMemory(address);
      case 0b10100000: return state.writeAccumToMemory(address);
      case 0b11000000: return state.callTo(address);
      default: throw new IllegalArgumentException();
    }
  }
}
```

You can use binary literals to make a bitmap more readable:

```
public static final short[] HAPPY_FACE = {
   (short)0b0000011111100000;
   (short)0b0000100000010000;
   (short)0b0001000000001000;
   (short)0b0010000000000100;
   (short)0b0100000000000010;
```

```
    (short)0b1000011001100001;
    (short)0b1000011001100001;
    (short)0b1000000000000001;
    (short)0b1000000000000001;
    (short)0b1001000000001001;
    (short)0b1000100000010001;
    (short)0b0100011111100010;
    (short)0b0010000000000100;
    (short)0b0001000000001000;
    (short)0b0000100000010000;
    (short)0b0000011111100000;
}
```


Contact Us