ref

June 14, 2022

1 Java References

[15]:
```
import java.lang.*
```

1.1 Basic Data Types

1.1.1 Bytes

[34]:
```
public class Bytes {
    public static void main(String[] args) {
        System.out.println("Number of bytes to represent a byte value: " + Byte.BYTES);
        System.out.println("Number of bits to represent a byte value: " + Byte.SIZE);
        System.out.println("Min: " + Byte.MIN_VALUE);
        System.out.println("Max: " + Byte.MAX_VALUE);
        System.out.println("Type: " + Byte.TYPE);
        System.out.println();

        // No default constructors
        byte first = Byte.parseByte("-123");
        byte second = Byte.parseByte("127");

        System.out.println("Parsing -123 and 127 into bytes: " + first + ", " +␣
↪second);
        System.out.println("Decode String to byte: " + Byte.decode("69"));
        System.out.println("Byte Value of first: " + first);
        System.out.println("Byte Value of second: " + second);
        System.out.println("Compare first with second: " + Byte.compare(first,␣
↪second));
        System.out.println("Comparing first to second (first, second are unsigned): " +
                Byte.compareUnsigned(first, second));
        System.out.println("Unsigned Int Value: " + Byte.toUnsignedInt(first));
        System.out.println("Unsigned Float Value: " + Byte.toUnsignedLong(first));
        System.out.println("String Value: " + Byte.toString(first));
        System.out.println("Value of first, second: " + Byte.valueOf(first) + ", " +␣
↪Byte.valueOf(second));
        System.out.println("Hashcode of first: " + Byte.hashCode(first));
        System.out.println();

        // Casting
        System.out.println("Cast to Int: " + (int) first);
        System.out.println("Cast to Long: " + (long) first);
        System.out.println("Cast to Short: " + (short) first);
```

```
        System.out.println("Cast to Double: " + (double) first);
        System.out.println("Cast to Float: " + (float) first);
        System.out.println("Cast to String: " + first);
        System.out.println();

        // Comparator
        System.out.println("<: " + (first < second));
        System.out.println(">: " + (first > second));
        System.out.println("<=: " + (first <= second));
        System.out.println(">=: " + (first >= second));
        System.out.println("!=: " + (first != second));
        System.out.println();

        // Unary Operators
        System.out.println("&: " + (first & second));
        System.out.println("|: " + (first | second));
        System.out.println();

        // Operators
        System.out.println("+: " + (first + second));
        System.out.println("-: " + (first - second));
        System.out.println("*: " + (first * second));
        System.out.println("/: " + (first / second));
        System.out.println("%: " + (first % second));
    }
}


Bytes bytes = new Bytes();
bytes.main(new String[1]);
```

```
Number of bytes to represent a byte value: 1
Number of bits to represent a byte value: 8
Min: -128
Max: 127
Type: byte

Parsing -123 and 127 into bytes: -123, 127
Decode String to byte: 69
Byte Value of first: -123
Byte Value of second: 127
Compare first with second: -250
Comparing first to second (first, second are unsigned): 6
Unsigned Int Value: 133
Unsigned Float Value: 133
String Value: -123
Value of first, second: -123, 127
Hashcode of first: -123

Cast to Int: -123
Cast to Long: -123
Cast to Short: -123
Cast to Double: -123.0
Cast to Float: -123.0
```

```
Cast to String: -123

<: true
>: false
<=: true
>=: false
!=: true

&: 5
|: -1

+: 4
-: -250
*: -15621
/: 0
%: -123
```

1.1.2 Boolean

[35]:
```
public class Bool {
    public static void main(String[] args) {
        boolean False = false;
        boolean True = true;

        System.out.println("FALSE Value: " + Boolean.FALSE);
        System.out.println("TRUE Value: " + Boolean.TRUE);
        System.out.println("Type: " + Boolean.TYPE);
        System.out.println();

        System.out.println("Parse String to Boolean: " + Boolean.parseBoolean("false")␣
↪+ ", " +
                Boolean.parseBoolean("true"));
        System.out.println("Compare booleans: " + Boolean.compare(False, True));
        System.out.println("Logical And: " + Boolean.logicalAnd(False, True));
        System.out.println("Logical Or: " + Boolean.logicalOr(False, True));
        System.out.println("Logical Exclusive Or: " + Boolean.logicalXor(False, True));
        System.out.println("Returns value of boolean: " + Boolean.valueOf(False) + ",␣
↪" + Boolean.valueOf(True));
        System.out.println("Returns boolean representation of string value: " +
                Boolean.valueOf("false") + ", " + Boolean.valueOf("true"));
        System.out.println("Convert to String: " + Boolean.toString(False) + ", " +␣
↪Boolean.toString(True));
        System.out.println("False, True hashcodes: " + Boolean.hashCode(False) + ", "␣
↪+ Boolean.hashCode(True));
        System.out.println();

        // Casting
        System.out.println("Cast to String: " + False);
        System.out.println();

        // Comparator
        System.out.println("!=: " + (False != True));
        System.out.println("&& : " + (False && True));
```

```
        System.out.println("|| : " + (False || True));
        System.out.println();

        // Unary Operators
        System.out.println("&: " + (False & True));
        System.out.println("|: " + (False | True));
        System.out.println("^: " + (False ^ True));
        System.out.println();
    }
}

Bool bool = new Bool();
bool.main(new String[1]);
```

```
FALSE Value: false
TRUE Value: true
Type: boolean

Parse String to Boolean: false, true
Compare booleans: -1
Logical And: false
Logical Or: true
Logical Exclusive Or: true
Returns value of boolean: false, true
Returns boolean representation of string value: false, true
Convert to String: false, true
False, True hashcodes: 1237, 1231

Cast to String: false

!=: true
&& : false
|| : true

&: false
|: true
^: true
```

1.1.3 Short

[37]:
```
public class Shorts {
    public static void main(String[] args) {
        System.out.println("Number of bytes to represent a short value: " + Short.
 ↪BYTES);
        System.out.println("Number of bits to represent a short value: " + Short.SIZE);
        System.out.println("Min: " + Short.MIN_VALUE);
        System.out.println("Max: " + Short.MAX_VALUE);
        System.out.println("Type: " + Short.TYPE);
        System.out.println();

        // No default constructors
        short first = Short.parseShort("-21");
```

```
        short second = Short.parseShort("12");

        System.out.println("Parsing strings -21 and 12 into short: " + first + ", " +␣
↪second);
        System.out.println("Decode String to short: " + Short.decode("69"));
        System.out.println("Short Value of first: " + first);
        System.out.println("Short Value of second: " + second);
        System.out.println("Compare second with first: " + Short.compare(second,␣
↪first));
        // Overflow occurs since it is unsigned
        System.out.println("Comparing first to second (first, second are unsigned): " +
                Short.compareUnsigned(first, second));
        // Overflow occurs since it is unsigned
        System.out.println("Unsigned Int Value: " + Short.toUnsignedInt(first));
        System.out.println("Unsigned Float Value: " + Short.toUnsignedLong(first));
        System.out.println("String Value: " + Short.toString(first));
        System.out.println("Value of first, second: " + Short.valueOf(first) + ", " +␣
↪Short.valueOf(second));
        System.out.println("Value converted from string: " + Short.valueOf("123"));
        System.out.println("Hashcode of first: " + Short.hashCode(first));
        System.out.println("Reverse the order of the bytes in the two complement␣
↪representation of the " +
                "short value: " + Short.reverseBytes(first));
        System.out.println();

        // Casting
        System.out.println("Cast to Int: " + (int) first);
        System.out.println("Cast to Long: " + (long) first);
        System.out.println("Cast to Double: " + (double) first);
        System.out.println("Cast to Float: " + (float) first);
        System.out.println("Cast to String: " + first);
        System.out.println();

        // Comparator
        System.out.println("<: " + (first < second));
        System.out.println(">: " + (first > second));
        System.out.println("<=: " + (first <= second));
        System.out.println(">=: " + (first >= second));
        System.out.println("!=: " + (first != second));
        System.out.println();

        // Unary Operators
        System.out.println("&: " + (first & second));
        System.out.println("|: " + (first | second));
        System.out.println("^: " + (first ^ second));
        System.out.println();

        // Operators
        System.out.println("+: " + (first + second));
        System.out.println("-: " + (first - second));
        System.out.println("*: " + (first * second));
        System.out.println("/: " + (first / second));
        System.out.println("%: " + (first % second));
```

```
    }
}

Shorts shorts = new Shorts();
shorts.main(new String[1]);
```

```
Number of bytes to represent a short value: 2
Number of bits to represent a short value: 16
Min: -32768
Max: 32767
Type: short

Parsing strings -21 and 12 into short: -21, 12
Decode String to short: 69
Short Value of first: -21
Short Value of second: 12
Compare second with first: 33
Comparing first to second (first, second are unsigned): 65503
Unsigned Int Value: 65515
Unsigned Float Value: 65515
String Value: -21
Value of first, second: -21, 12
Value converted from string: 123
Hashcode of first: -21
Reverse the order of the bytes in the two complement representation of the short
value: -5121

Cast to Int: -21
Cast to Long: -21
Cast to Double: -21.0
Cast to Float: -21.0
Cast to String: -21

<: true
>: false
<=: true
>=: false
!=: true

&: 8
|: -17
^: -25

+: -9
-: -33
*: -252
/: -1
%: -9
```

1.1.4 Integer

[1]:
```
public class Ints {
    public static void main(String[] args) {
        System.out.println("Number of bytes to represent an int value: " + Integer.
↪BYTES);
        System.out.println("Number of bits to represent an int value: " + Integer.
↪SIZE);
        System.out.println("Min: " + Integer.MIN_VALUE);
        System.out.println("Max: " + Integer.MAX_VALUE);
        System.out.println("Type: " + Integer.TYPE);
        System.out.println();

        // No default constructors
        int first = 123;
        int second = -12;

        System.out.println("Parse Int: " + Integer.parseInt("6969"));
        System.out.println("Parse unsigned Int: " + Integer.parseUnsignedInt("928"));
        System.out.println("Count number of bits used to represent first: " + Integer.
↪bitCount(first));
        System.out.println("Compare second with first: " + Integer.compare(second,␣
↪first));
        System.out.println("Comparing first to second (first, second are unsigned): " +
                Integer.compareUnsigned(first, second));
        System.out.println("Decode String to short: " + Integer.decode("69"));
        System.out.println("Divide 2 integers and return unsigned quotient: " +␣
↪Integer.divideUnsigned(first, second));
        System.out.println("Divide 2 integers and return unsigned remainder: " +␣
↪Integer.remainderUnsigned(first, second));
        System.out.println("Lowest One bit (rightmost bit): " + Integer.
↪lowestOneBit(first));
        System.out.println("Highest One bit (rightmost bit): " + Integer.
↪highestOneBit(first));
        System.out.println("Min between 2 ints: " + Integer.min(first, second));
        System.out.println("Max between 2 ints: " + Integer.max(first, second));
        System.out.println("Number of leading zeros: " + Integer.
↪numberOfLeadingZeros(first));
        System.out.println("Number of trailing zeros: " + Integer.
↪numberOfTrailingZeros(first));
        System.out.println("Reverse the order of the bits in the two complement␣
↪representation of the " +
                "int value: " + Integer.reverse(first));
        System.out.println("Reverse the order of the bytes in the two complement␣
↪representation of the " +
                "int value: " + Integer.reverseBytes(first));
        System.out.println("Rotate (left) 2 complementary binary representation of int␣
↪by number of bits: " +
                Integer.rotateLeft(first, 1));
        System.out.println("Rotate (left) 2 complementary binary representation of int␣
↪by number of bits: " +
                Integer.rotateRight(first, 1));
```

```
        System.out.println("Signum function of int value: " + Integer.signum(first));
        System.out.println("Sum of 2 Integers: " + Integer.sum(first, second));
        System.out.println("Binary String: " + Integer.toBinaryString(first));
        System.out.println("Hex String: " + Integer.toHexString(first));
        System.out.println("Octal String: " + Integer.toOctalString(first));
        System.out.println("String Value: " + Integer.toBinaryString(first));
        System.out.println("Unsigned Long Value: " + Integer.toUnsignedLong(first));
        System.out.println("Unsigned String Value: " + Integer.
↪toUnsignedString(first));
        System.out.println("Value of first, second: " + Integer.valueOf(first) + ", "␣
↪+ Integer.valueOf(second));
        System.out.println("Value converted from string: " + Integer.valueOf("123"));
        System.out.println("Hashcode of first: " + Integer.hashCode(first));

        System.out.println();

        // Casting
        System.out.println("Cast to Long: " + (long) first);
        System.out.println("Cast to Double: " + (double) first);
        System.out.println("Cast to Short: " + (short) first);
        System.out.println("Cast to Float: " + (float) first);
        System.out.println("Cast to Byte: " + (byte) first);
        System.out.println("Cast to String: " + first);
        System.out.println();

        // Comparator
        System.out.println("<: " + (first < second));
        System.out.println(">: " + (first > second));
        System.out.println("<=: " + (first <= second));
        System.out.println(">=: " + (first >= second));
        System.out.println("!=: " + (first != second));
        System.out.println();

        // Unary Operators
        System.out.println("&: " + (first & second));
        System.out.println("|: " + (first | second));
        System.out.println("^: " + (first ^ second));
        System.out.println();

        // Operators
        System.out.println("+: " + (first + second));
        System.out.println("-: " + (first - second));
        System.out.println("*: " + (first * second));
        System.out.println("/: " + (first / second));
        System.out.println("%: " + (first % second));
    }
}

Ints ints = new Ints();
ints.main(new String[1]);
```

```
Number of bytes to represent an int value: 4
Number of bits to represent an int value: 32
```

```
Min: -2147483648
Max: 2147483647
Type: int

Parse Int: 6969
Parse unsigned Int: 928
Count number of bits used to represent first: 6
Compare second with first: -1
Comparing first to second (first, second are unsigned): -1
Decode String to short: 69
Divide 2 integers and return unsigned quotient: 0
Divide 2 integers and return unsigned remainder: 123
Lowest One bit (rightmost bit): 1
Highest One bit (rightmost bit): 64
Min between 2 ints: -12
Max between 2 ints: 123
Number of leading zeros: 25
Number of trailing zeros: 0
Reverse the order of the bits in the two complement representation of the int
value: -570425344
Reverse the order of the bytes in the two complement representation of the int
value: 2063597568
Rotate (left) 2 complementary binary representation of int by number of bits:
246
Rotate (left) 2 complementary binary representation of int by number of bits:
-2147483587
Signum function of int value: 1
Sum of 2 Integers: 111
Binary String: 1111011
Hex String: 7b
Octal String: 173
String Value: 1111011
Unsigned Long Value: 123
Unsigned String Value: 123
Value of first, second: 123, -12
Value converted from string: 123
Hashcode of first: 123

Cast to Long: 123
Cast to Double: 123.0
Cast to Short: 123
Cast to Float: 123.0
Cast to Byte: 123
Cast to String: 123

<: false
>: true
<=: false
>=: true
!=: true

&: 112
|: -1
^: -113
```

```
+: 111
-: 135
*: -1476
/: -10
%: 3
```

1.1.5 Long

[2]:
```
public class Longs {
    public static void main(String[] args) {
        System.out.println("Number of bytes to represent a long value: " + Long.BYTES);
        System.out.println("Number of bits to represent a long value: " + Long.SIZE);
        System.out.println("Min: " + Long.MIN_VALUE);
        System.out.println("Max: " + Long.MAX_VALUE);
        System.out.println("Type: " + Long.TYPE);
        System.out.println();

        // No default constructors
        long first = 123423423;
        long second = -12344232;

        System.out.println("Parse Long: " + Long.parseLong("6969"));
        System.out.println("Parse unsigned Long: " + Long.parseUnsignedLong("928"));
        System.out.println("Count number of bits used to represent first: " + Long.
↪bitCount(first));
        System.out.println("Compare second with first: " + Long.compare(second,␣
↪first));
        System.out.println("Comparing first to second (first, second are unsigned): " +
                Long.compareUnsigned(first, second));
        System.out.println("Decode String to short: " + Long.decode("-2342342369"));
        System.out.println("Divide 2 integers and return unsigned quotient: " + Long.
↪divideUnsigned(first, second));
        System.out.println("Divide 2 integers and return unsigned remainder: " + Long.
↪remainderUnsigned(first, second));
        System.out.println("Lowest One bit (rightmost bit): " + Long.
↪lowestOneBit(first));
        System.out.println("Highest One bit (rightmost bit): " + Long.
↪highestOneBit(first));
        System.out.println("Min between 2 longs: " + Long.min(first, second));
        System.out.println("Max between 2 longs: " + Long.max(first, second));
        System.out.println("Number of leading zeros: " + Long.
↪numberOfLeadingZeros(first));
        System.out.println("Number of trailing zeros: " + Long.
↪numberOfTrailingZeros(first));
        System.out.println("Reverse the order of the bits in the two complement␣
↪representation of the " +
                "int value: " + Long.reverse(first));
        System.out.println("Reverse the order of the bytes in the two complement␣
↪representation of the " +
                "int value: " + Long.reverseBytes(first));
        System.out.println("Rotate (left) 2 complementary binary representation of int␣
↪by number of bits: " +
```

```
                Long.rotateLeft(first, 1));
        System.out.println("Rotate (left) 2 complementary binary representation of int␣
↪by number of bits: " +
                Long.rotateRight(first, 1));
        System.out.println("Signum function of int value: " + Long.signum(first));
        System.out.println("Sum of 2 Longs: " + Long.sum(first, second));
        System.out.println("Binary String: " + Long.toBinaryString(first));
        System.out.println("Hex String: " + Long.toHexString(first));
        System.out.println("Octal String: " + Long.toOctalString(first));
        System.out.println("String Value: " + Long.toBinaryString(first));
        System.out.println("Unsigned String Value: " + Long.toUnsignedString(first));
        System.out.println("Value of first, second: " + Long.valueOf(first) + ", " +␣
↪Long.valueOf(second));
        System.out.println("Value converted from string: " + Long.valueOf("123"));
        System.out.println("Hashcode of first: " + Long.hashCode(first));
        System.out.println();

        // Casting
        System.out.println("Cast to Integer: " + (int) first);
        System.out.println("Cast to Double: " + (double) first);
        System.out.println("Cast to Short: " + (short) first);
        System.out.println("Cast to Float: " + (float) first);
        System.out.println("Cast to Byte: " + (byte) first);
        System.out.println("Cast to String: " + first);
        System.out.println();

        // Comparator
        System.out.println("<: " + (first < second));
        System.out.println(">: " + (first > second));
        System.out.println("<=: " + (first <= second));
        System.out.println(">=: " + (first >= second));
        System.out.println("!=: " + (first != second));
        System.out.println();

        // Unary Operators
        System.out.println("&: " + (first & second));
        System.out.println("|: " + (first | second));
        System.out.println("^: " + (first ^ second));
        System.out.println();

        // Operators
        System.out.println("+: " + (first + second));
        System.out.println("-: " + (first - second));
        System.out.println("*: " + (first * second));
        System.out.println("/: " + (first / second));
        System.out.println("%: " + (first % second));
    }
}

Longs longs = new Longs();
longs.main(new String[1]);
```

```
Number of bytes to represent a long value: 8
```

```
Number of bits to represent a long value: 64
Min: -9223372036854775808
Max: 9223372036854775807
Type: long

Parse Long: 6969
Parse unsigned Long: 928
Count number of bits used to represent first: 18
Compare second with first: -1
Comparing first to second (first, second are unsigned): -1
Decode String to short: -2342342369
Divide 2 integers and return unsigned quotient: 0
Divide 2 integers and return unsigned remainder: 123423423
Lowest One bit (rightmost bit): 1
Highest One bit (rightmost bit): 67108864
Min between 2 longs: -12344232
Max between 2 longs: 123423423
Number of leading zeros: 37
Number of trailing zeros: 0
Reverse the order of the bits in the two complement representation of the int
value: -192851178415980544
Reverse the order of the bytes in the two complement representation of the int
value: -4662814378565828608
Rotate (left) 2 complementary binary representation of int by number of bits:
246846846
Rotate (left) 2 complementary binary representation of int by number of bits:
-9223372036793064097
Signum function of int value: 1
Sum of 2 Longs: 111079191
Binary String: 11101011011010010101010111111
Hex String: 75b4abf
Octal String: 726645277
String Value: 11101011011010010101010111111
Unsigned String Value: 123423423
Value of first, second: 123423423, -12344232
Value converted from string: 123
Hashcode of first: 123423423

Cast to Integer: 123423423
Cast to Double: 1.23423423E8
Cast to Short: 19135
Cast to Float: 1.23423424E8
Cast to Byte: -65
Cast to String: 123423423

<: false
>: true
<=: false
>=: true
!=: true

&: 121831448
|: -10752257
^: -132583705
```

```
+: 111079191
-: 135767655
*: -1523567367746136
/: -9
%: 12325335
```

1.1.6 Float

[3]:
```java
public class Floats {
    public static void main(String[] args) {
        System.out.println("Number of bytes to represent a float value: " + Float.
 ↪BYTES);
        System.out.println("Number of bits to represent a float value: " + Float.SIZE);
        System.out.println("Min: " + Float.MIN_VALUE);
        System.out.println("Max: " + Float.MAX_VALUE);
        System.out.println("Smallest value of normal float: " + Float.MIN_NORMAL);
        System.out.println("Min Exponent: " + Float.MIN_EXPONENT);
        System.out.println("Max Exponent: " + Float.MAX_EXPONENT);
        System.out.println("NaN: " + Float.NaN);
        System.out.println("Neg Inf: " + Float.NEGATIVE_INFINITY);
        System.out.println("Pos Inf: " + Float.POSITIVE_INFINITY);
        System.out.println("Type: " + Float.TYPE);
        System.out.println();

        // No default constructors
        float first = 123.213f;
        float second = -12344.237f;

        System.out.println("Parse Long: " + Float.parseFloat("6969.69696"));
        System.out.println("Count number of bits used to represent first as int: " +␣
 ↪Float.floatToIntBits(first));
        System.out.println("Count number of bits used to represent first as raw int: "␣
 ↪+ Float.floatToRawIntBits(first));
        System.out.println("Convert int bits to float: " + Float.intBitsToFloat(100));
        System.out.println("Compare second with first: " + Float.compare(second,␣
 ↪first));
        System.out.println("Comparing first to second: " + Float.compare(first,␣
 ↪second));
        System.out.println("Check if is infinite: " + Float.isInfinite(first));
        System.out.println("Check if is finite: " + Float.isFinite(first));
        System.out.println("Check if is NaN: " + Float.isNaN(first));
        System.out.println("Min between 2 float: " + Float.min(first, second));
        System.out.println("Max between 2 float: " + Float.max(first, second));
        System.out.println("Sum of 2 Floats: " + Float.sum(first, second));
        System.out.println("Hex String: " + Float.toHexString(first));
        System.out.println("String value: " + Float.toString(first));
        System.out.println("Value of first, second: " + Float.valueOf(first) + ", " +␣
 ↪Float.valueOf(second));
        System.out.println("Value converted from string: " + Float.valueOf("123"));
        System.out.println("Hashcode of first: " + Float.hashCode(first));
        System.out.println();
```

```
        // Casting
        System.out.println("Cast to Integer: " + (int) first);
        System.out.println("Cast to Double: " + (double) first);
        System.out.println("Cast to Short: " + (short) first);
        System.out.println("Cast to Long: " + (long) first);
        System.out.println("Cast to Byte: " + (byte) first);
        System.out.println("Cast to String: " + first);
        System.out.println();

        // Comparator
        System.out.println("<: " + (first < second));
        System.out.println(">: " + (first > second));
        System.out.println("<=: " + (first <= second));
        System.out.println(">=: " + (first >= second));
        System.out.println("!=: " + (first != second));
        System.out.println();

        // Operators
        System.out.println("+: " + (first + second));
        System.out.println("-: " + (first - second));
        System.out.println("*: " + (first * second));
        System.out.println("/: " + (first / second));
        System.out.println("%: " + (first % second));
    }
}

Floats floats = new Floats();
floats.main(new String[1]);
```

```
Number of bytes to represent a float value: 4
Number of bits to represent a float value: 32
Min: 1.4E-45
Max: 3.4028235E38
Smallest value of normal float: 1.17549435E-38
Min Exponent: -126
Max Exponent: 127
NaN: NaN
Neg Inf: -Infinity
Pos Inf: Infinity
Type: float

Parse Long: 6969.697
Count number of bits used to represent first as int: 1123446030
Count number of bits used to represent first as raw int: 1123446030
Convert int bits to float: 1.4E-43
Compare second with first: -1
Comparing first to second: 1
Check if is infinite: false
Check if is finite: true
Check if is NaN: false
Min between 2 float: -12344.237
Max between 2 float: 123.213
Sum of 2 Floats: -12221.024
Hex String: 0x1.ecda1cp6
```

```
String value: 123.213
Value of first, second: 123.213, -12344.237
Value converted from string: 123.0
Hashcode of first: 1123446030

Cast to Integer: 123
Cast to Double: 123.21299743652344
Cast to Short: 123
Cast to Long: 123
Cast to Byte: 123
Cast to String: 123.213

<: false
>: true
<=: false
>=: true
!=: true

+: -12221.024
-: 12467.45
*: -1520970.5
/: -0.009981418
%: 123.213
```

1.1.7 Double

[5]:
```
public class Doubles {
    public static void main(String[] args) {
        System.out.println("Number of bytes to represent a double value: " + Double.
 ↪BYTES);
        System.out.println("Number of bits to represent a double value: " + Double.
 ↪SIZE);
        System.out.println("Min: " + Double.MIN_VALUE);
        System.out.println("Max: " + Double.MAX_VALUE);
        System.out.println("Smallest value of normal double: " + Double.MIN_NORMAL);
        System.out.println("Min Exponent: " + Double.MIN_EXPONENT);
        System.out.println("Max Exponent: " + Double.MAX_EXPONENT);
        System.out.println("NaN: " + Double.NaN);
        System.out.println("Neg Inf: " + Double.NEGATIVE_INFINITY);
        System.out.println("Pos Inf: " + Double.POSITIVE_INFINITY);
        System.out.println("Type: " + Double.TYPE);
        System.out.println();

        // No default constructors
        double first = 123.232452345454313d;
        double second = -12344534.23453453437d;

        System.out.println("Parse Long: " + Double.parseDouble("-6969.69696"));
        System.out.println("Count number of bits used to represent first as long: " +␣
 ↪Double.doubleToLongBits(first));
        System.out.println("Count number of bits used to represent first as raw long:␣
 ↪" + Double.doubleToRawLongBits(first));
```

```
        System.out.println("Convert long bits to double: " + Double.
↪longBitsToDouble(100));
        System.out.println("Compare second with first: " + Double.compare(second,␣
↪first));
        System.out.println("Comparing first to second: " + Double.compare(first,␣
↪second));
        System.out.println("Check if is infinite: " + Double.isInfinite(first));
        System.out.println("Check if is finite: " + Double.isFinite(first));
        System.out.println("Check if is NaN: " + Double.isNaN(first));
        System.out.println("Min between 2 double: " + Double.min(first, second));
        System.out.println("Max between 2 double: " + Double.max(first, second));
        System.out.println("Sum of 2 doubles: " + Double.sum(first, second));
        System.out.println("Hex String: " + Double.toHexString(first));
        System.out.println("String value: " + Double.toString(first));
        System.out.println("Value of first, second: " + Double.valueOf(first) + ", " +␣
↪Double.valueOf(second));
        System.out.println("Value converted from string: " + Double.valueOf("123"));
        System.out.println("Hashcode of first: " + Double.hashCode(first));
        System.out.println();

        // Casting
        System.out.println("Cast to Integer: " + (int) first);
        System.out.println("Cast to Float: " + (float) first);
        System.out.println("Cast to Short: " + (short) first);
        System.out.println("Cast to Long: " + (long) first);
        System.out.println("Cast to Byte: " + (byte) first);
        System.out.println("Cast to String: " + first);
        System.out.println();

        // Comparator
        System.out.println("<: " + (first < second));
        System.out.println(">: " + (first > second));
        System.out.println("<=: " + (first <= second));
        System.out.println(">=: " + (first >= second));
        System.out.println("!=: " + (first != second));
        System.out.println();

        // Operators
        System.out.println("+: " + (first + second));
        System.out.println("-: " + (first - second));
        System.out.println("*: " + (first * second));
        System.out.println("/: " + (first / second));
        System.out.println("%: " + (first % second));
    }
}

Doubles doubles = new Doubles();
doubles.main(new String[1]);
```

```
Number of bytes to represent a double value: 8
Number of bits to represent a double value: 64
Min: 4.9E-324
Max: 1.7976931348623157E308
```

```
Smallest value of normal double: 2.2250738585072014E-308
Min Exponent: -1022
Max Exponent: 1023
NaN: NaN
Neg Inf: -Infinity
Pos Inf: Infinity
Type: double

Parse Long: -6969.69696
Count number of bits used to represent first as long: 4638372129850353333
Count number of bits used to represent first as raw long: 4638372129850353333
Convert long bits to double: 4.94E-322
Compare second with first: -1
Comparing first to second: 1
Check if is infinite: false
Check if is finite: true
Check if is NaN: false
Min between 2 double: -1.2344534234534534E7
Max between 2 double: 123.23245234545432
Sum of 2 doubles: -1.2344411002082188E7
Hex String: 0x1.ecee07fcd66b5p6
String value: 123.23245234545432
Value of first, second: 123.23245234545432, -1.2344534234534534E7
Value converted from string: 123.0
Hashcode of first: 1066641493

Cast to Integer: 123
Cast to Float: 123.23245
Cast to Short: 123
Cast to Long: 123
Cast to Byte: 123
Cast to String: 123.23245234545432

<: false
>: true
<=: false
>=: true
!=: true

+: -1.2344411002082188E7
-: 1.234465746698688E7
*: -1.5212472267841063E9
/: -9.982754310867764E-6
%: 123.23245234545432
```

1.1.8 Char

[4]:
```java
public class Chars {
    public static void main(String[] args) {
        // Other stuff about unicode specs is not included
        System.out.println("Number of bytes to represent a char value: " + Character.
 ↪BYTES);
```

```
        System.out.println("Number of bits to represent a char value: " + Character.
↪SIZE);
        // System.out.println("Min: " + Character.MIN_VALUE); => unprintable
        // System.out.println("Max: " + Character.MAX_VALUE); => unprintable
        System.out.println("Type: " + Character.TYPE);
        System.out.println("Max Radix: " + Character.MAX_RADIX);
        System.out.println("Min Radix: " + Character.MIN_RADIX);
        System.out.println();

        char first = 'A';
        char second = 'z';
        char[] third = new char[]{'A', 'b', 'c', '?'};

        System.out.println("Number of char values represent char: " + Character.
↪charCount(first));
        System.out.println("Code point from char: " + Character.toCodePoint(first,␣
↪second));
        System.out.println("Code point (ASCII value) in char array: " + Character.
↪codePointAt(third, 0));
        System.out.println("Code point at (ASCII value) in char array with limits: " +␣
↪Character.codePointAt(third, 0, 1));
        System.out.println("Code point before (ASCII value) in char array with limits:␣
↪" + Character.codePointBefore(third, 2));
        System.out.println("Code point count (ASCII value) in char array: " +␣
↪Character.codePointCount(third, 0, 1));
        System.out.println("Compare chars: " + Character.compare(first, second));
        System.out.println("Digit (Numeric value of char in specific radix): " +␣
↪Character.digit(first, 0));
        // System.out.println("Digit (Code Point in specific radix): " + Character.
↪digit(65, 0));
        // System.out.println("Char representation of specific digit in radix: " +␣
↪Character.forDigit(65, 0));
        System.out.println("Directional Property of Char: " + Character.
↪getDirectionality(first));
        System.out.println("Directional Property of Code point: " + Character.
↪getDirectionality(65));
        System.out.println("Numerical Value of char: " + Character.
↪getNumericValue(first));
        System.out.println("Numerical Value of code point: " + Character.
↪getNumericValue(12));
        System.out.println("Type of char: " + Character.getType(first));
        System.out.println("Type of code point: " + Character.getType(65));
        System.out.println("Hashcode: " + Character.hashCode(first));
        System.out.println();

        // all works with code points idk
        System.out.println("Is Alphabetic: " + Character.isAlphabetic(first));
        System.out.println("Is Digit: " + Character.isDigit(first));
        System.out.println("Is Letter: " + Character.isLetter(first));
        System.out.println("Is Letter or Digit: " + Character.isLetterOrDigit(first));
        System.out.println("Is Lower Case: " + Character.isLowerCase(first));
        System.out.println("To Lower Case: " + Character.toLowerCase(first));
```

```
        System.out.println("Is Title Case: " + Character.isTitleCase(first));
        System.out.println("To Title Case: " + Character.toTitleCase(first));
        System.out.println("Is Upper Case: " + Character.isUpperCase(first));
        System.out.println("To Upper Case: " + Character.toUpperCase(first));
        System.out.println("Is Space Char: " + Character.isSpaceChar(first));
        System.out.println("Is Whitespace Char: " + Character.isWhitespace(first));
        // System.out.println("Reverse Bytes: " + Character.reverseBytes(first));
        System.out.println("Convert to String: " + Character.toString(first));
        System.out.println("Value of: " + Character.valueOf(first));
        System.out.println();

        // Casting
        System.out.println("Cast to Int: " + (int) first);
        System.out.println("Cast to Long: " + (long) first);
        System.out.println("Cast to Double: " + (double) first);
        System.out.println("Cast to Float: " + (float) first);
        System.out.println("Cast to Byte: " + (byte) first);
        System.out.println("Cast to String: " + first);
        System.out.println();

        // Comparator
        System.out.println("<: " + (first < second));
        System.out.println(">: " + (first > second));
        System.out.println("<=: " + (first <= second));
        System.out.println(">=: " + (first >= second));
        System.out.println("!=: " + (first != second));
        System.out.println();

        // Unary Operators
        System.out.println("&: " + (first & second));
        System.out.println("|: " + (first | second));
        System.out.println("^: " + (first ^ second));
        System.out.println();

        // Operators
        System.out.println("+: " + (first + second));
        System.out.println("-: " + (first - second));
        System.out.println("*: " + (first * second));
        System.out.println("/: " + (first / second));
        System.out.println("%: " + (first % second));
    }
}

Chars chars = new Chars();
chars.main(new String[1]);
```

```
Number of bytes to represent a char value: 2
Number of bits to represent a char value: 16
Type: char
Max Radix: 36
Min Radix: 2

Number of char values represent char: 1
Code point from char: -56547206
```

```
Code point (ASCII value) in char array: 65
Code point at (ASCII value) in char array with limits: 65
Code point before (ASCII value) in char array with limits: 98
Code point count (ASCII value) in char array: 1
Compare chars: -57
Digit (Numeric value of char in specific radix): -1
Directional Property of Char: 0
Directional Property of Code point: 0
Numerical Value of char: 10
Numerical Value of code point: -1
Type of char: 1
Type of code point: 1
Hashcode: 65

Is Alphabetic: true
Is Digit: false
Is Letter: true
Is Letter or Digit: true
Is Lower Case: false
To Lower Case: a
Is Title Case: false
To Title Case: A
Is Upper Case: true
To Upper Case: A
Is Space Char: false
Is Whitespace Char: false
Convert to String: A
Value of: A

Cast to Int: 65
Cast to Long: 65
Cast to Double: 65.0
Cast to Float: 65.0
Cast to Byte: 65
Cast to String: A

<: true
>: false
<=: true
>=: false
!=: true

&: 64
|: 123
^: 59

+: 187
-: -57
*: 7930
/: 0
%: 65
```

1.1.9 String

[6]:
```
import java.nio.charset.StandardCharsets;

public class Strings {
    public static void main(String[] args) {
        // Constructors
        byte[] bytes = new byte[]{(byte) 1, (byte) 2};
        char[] chars = new char[]{'A', 'z'};
        int[] ints = new int[]{65, 64};

        String byteStr = new String(bytes);
        String charStr = new String(chars);
        String intStr = new String(ints, 0, 2);
        String normalString = "This is a normal String";

        // System.out.println("String from byte array: " + byteStr);
        System.out.println("String from char array: " + charStr);
        System.out.println("String from int array: " + intStr);
        System.out.println();

        // Methods
        System.out.println("Char at: " + normalString.charAt(2));
        System.out.println("Code point at: " + normalString.codePointAt(2));
        System.out.println("Code point before: " + normalString.codePointBefore(2));
        System.out.println("Code point count: " + normalString.codePointCount(0, 2));
        System.out.println("Compare to: " + normalString.compareTo(charStr));
        System.out.println("Compare to (ignore case): " + normalString.
↪compareToIgnoreCase(charStr));
        System.out.println("Concat strings: " + normalString.concat(charStr));
        System.out.println("Copy String: " + String.copyValueOf(new char[] {'A', 'B',␣
↪'C'}));
        System.out.println("Ends with: " + normalString.endsWith("g"));
        System.out.println("Starts with: " + normalString.startsWith("T"));
        System.out.println("Equals: " + normalString.equals(charStr));
        System.out.println("Equals (Ignore case): " + normalString.
↪equalsIgnoreCase(charStr));
        char[] temp = new char[10];
        normalString.getChars(0, 10, temp, 0);
        System.out.println("Get Chars: " + temp.toString());
        System.out.println("Index of: " + normalString.indexOf('T'));
        System.out.println("Last Index of (char): " + normalString.lastIndexOf('T'));
        System.out.println("Last Index of (string): " + normalString.
↪lastIndexOf("Th"));
        System.out.println("Index of (from index): " + normalString.indexOf('S', 9));
        System.out.println("Intern: " + normalString.intern());
        System.out.println("Is Blank: " + normalString.isBlank());
        System.out.println("Is Empty: " + normalString.isEmpty());
        System.out.println("Length: " + normalString.length());
        System.out.println("Offset by code points: " + normalString.
↪offsetByCodePoints(0, 9));
        System.out.println("Check if region of string are equal: " + normalString.
↪regionMatches(0, charStr, 0, 10));
```

```
        System.out.println("String duplication and concatenation (A * 5 in python): "␣
↪+ normalString.repeat(2));
        System.out.println("Replace char in string: " + normalString.replace('T',␣
↪'A'));
        System.out.println("Strip whitespaces: " + "    stripped      ".strip());
        System.out.println("Strip leading whitespaces: " + "    stripped      ".
↪stripLeading());
        System.out.println("Strip trailing whitespaces: " + "    stripped      ".
↪stripTrailing());
        System.out.println("Trim string (remove whitespaces and convert codepoints): "␣
↪+
                "    stripped      ".trim());
        System.out.println("Subsequence: " + normalString.subSequence(0, 10));
        System.out.println("Substring: " + normalString.substring(3));
        System.out.println("Substring: " + normalString.substring(0, 10));
        System.out.println("Convert to char array: " + normalString.toCharArray());
        System.out.println("Convert to lower case: " + normalString.toLowerCase());
        System.out.println("Convert to upper case: " + normalString.toUpperCase());
        System.out.println("Convert to string: " + normalString.toString());
        System.out.println();

        // Conversion
        System.out.println("Boolean: " + String.valueOf(false));
        System.out.println("Char: " + String.valueOf('A'));
        System.out.println("Char Array: " + String.valueOf(new char[]{'A', 'B'}));
        System.out.println("Double: " + String.valueOf(9213.3124d));
        System.out.println("Float: " + String.valueOf(123.34f));
        System.out.println("Integer: " + String.valueOf(123));
        System.out.println("Long: " + String.valueOf(12341423453254L));

        // Find hashcode
        System.out.println("Hashcode: " + normalString.hashCode());

        // Comparator (not a
        System.out.println("!=: " + (normalString != charStr));
        System.out.println();

        // Operators
        System.out.println("+: " + (normalString + charStr));
    }
}

Strings strings = new Strings();
strings.main(new String[1]);
```

```
String from char array: Az
String from int array: A@

Char at: i
Code point at: 105
Code point before: 104
Code point count: 2
Compare to: 19
```

```
Compare to (ignore case): 19
Concat strings: This is a normal StringAz
Copy String: ABC
Ends with: true
Starts with: true
Equals: false
Equals (Ignore case): false
Get Chars: [C@57c7de31
Index of: 0
Last Index of (char): 0
Last Index of (string): 0
Index of (from index): 17
Intern: This is a normal String
Is Blank: false
Is Empty: false
Length: 23
Offset by code points: 9
Check if region of string are equal: false
String duplication and concatenation (A * 5 in python): This is a normal
StringThis is a normal String
Replace char in string: Ahis is a normal String
Strip whitespaces: stripped
Strip leading whitespaces: stripped
Strip trailing whitespaces:     stripped
Trim string (remove whitespaces and convert codepoints): stripped
Subsequence: This is a
Substring: s is a normal String
Substring: This is a
Convert to char array: [C@59003ab9
Convert to lower case: this is a normal string
Convert to upper case: THIS IS A NORMAL STRING
Convert to string: This is a normal String

Boolean: false
Char: A
Char Array: AB
Double: 9213.3124
Float: 123.34
Integer: 123
Long: 12341423453254
Hashcode: -1745751625
!=: true

+: This is a normal StringAz
```

1.2 Array

[9]:
```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;

public class ArrayThings {
    public static void main(String[] args) {
```

```java
        // Constructor
        int[] intArray = new int[] {1, 2, 3, 4};
        int[] intArray2 = new int[] {1, 2, 3, 4};
        int[][] nestedIntArray = new int[][] {{1, 2}, {3, 4}};

        // Printing array
        System.out.println("Integer Array: " + Arrays.toString(intArray));
        System.out.println("Deep Integer Array:" + Arrays.
↪deepToString(nestedIntArray));

        // Length of array
        System.out.println("Length of array: " + intArray.length);

        // Indexing
        System.out.println("Index 0 of intArray: " + intArray[0]);

        // Setting value
        intArray[0] = 99999;
        System.out.println("Set index 0 of intArray: " + Arrays.toString(intArray));

        // Swapping value
        int temp = 0;
        temp = intArray[0];
        intArray[0] = intArray[1];
        intArray[1] = temp;
        System.out.println("Swapped: " + Arrays.toString(intArray));

        // Equals
        System.out.println("Equal Arrays (Shallow equality): " + Arrays.
↪equals(intArray, intArray2));
        System.out.println("Equal Arrays (Deep equality): " + Arrays.
↪deepEquals(nestedIntArray, nestedIntArray));

        // Binary Search
        // Returns index of the search key, if it is contained in the array;␣
↪otherwise, (-(insertion point) - 1)
        System.out.println("BSearch (found): " + Arrays.binarySearch(intArray, 4));
        System.out.println("BSearch (not found): " + Arrays.binarySearch(intArray,␣
↪-999));

        // Compare arrays, must be same type
        System.out.println("Compare (similar): " + Arrays.compare(intArray, intArray));
        System.out.println("Compare unsigned (similar): " + Arrays.
↪compareUnsigned(intArray, intArray));
        System.out.println("Compare (different): " + Arrays.compare(intArray,␣
↪intArray2));
        System.out.println("Compare unsigned (different): " + Arrays.
↪compareUnsigned(intArray, intArray2));

        // Copy array
        System.out.println("Copy Entire Length: " + Arrays.toString(Arrays.
↪copyOf(intArray, intArray.length)));
```

```
        System.out.println("Copy Slice: " + Arrays.toString(Arrays.copyOf(intArray,␣
↪intArray.length / 2)));
        System.out.println("Copy Range of Arrays: " + Arrays.toString(Arrays.
↪copyOfRange(intArray, 0, 2)));

        // Fill Array
        int[] empty = new int[10];
        Arrays.fill(empty, 0);
        System.out.println("Fill: " + Arrays.toString(empty));

        // Mismatched Items
        System.out.println("Mismatch: " + Arrays.mismatch(intArray, intArray2));

        // Sort
        Arrays.sort(intArray);
        System.out.println("Sorted: " + Arrays.toString(intArray));

        Arrays.parallelSort(intArray);
        System.out.println("Parallel Sorted: " + Arrays.toString(intArray));

        // Array List
        System.out.println();
        ArrayListThing();
        System.out.println();
        LinkedListThing();
    }

    public static void ArrayListThing() {
        // Integer Array
        ArrayList<Integer> arr = new ArrayList<Integer>();
        ArrayList<Integer> arr2 = new ArrayList<Integer>();

        // Add item
        arr.add(9);
        arr2.add(12);
        arr2.add(123124);

        // Length
        System.out.println("Array size: " + arr.size() + ", " + arr2.size());

        // Print
        System.out.println("Print: " + arr.toString() + ", " + arr2.toString());

        // Add all
        arr.addAll(arr2);
        System.out.println("Add from arr2 to arr: " + arr.toString() + ", " + arr2.
↪toString());

        // Clear
        arr2.clear();
        System.out.println("Clear: " + arr.toString() + ", " + arr2.toString());

        // Shallow copy
```

```
        System.out.println("Shallow copy: " + arr.clone().toString());

        // Contains
        System.out.println("Contains 1: " + arr.contains(1));

        // Get by Index
        System.out.println("Get 0: " + arr.get(0));

        // Check empty
        System.out.println("Empty: " + arr2.isEmpty());

        // Remove
        arr.remove(0);
        System.out.println("Remove by index: " + arr.toString());
        arr.remove((Integer) 123124);
        System.out.println("Remove by object: " + arr.toString());
        arr.removeAll(arr2);
        System.out.println("Remove all items that are contained in other sequence: " +␣
↪arr.toString());

        // Retain
        arr.retainAll(arr2);
        System.out.println("Keep items that are contained in the other sequence: " +␣
↪arr.toString());

        // Increase defined array size
        arr.ensureCapacity(1000);
        System.out.println("Increased max size: " + arr.toString());

        // Trim the list to current size
        arr.trimToSize();
        System.out.println("Trimmed to size: " + arr.toString());

        // Convert to array
        ArrayList<Integer> new_ = new ArrayList<Integer>();
        new_.add(123);
        new_.add(123);
        new_.add(21312);
        Object[] conv = new_.toArray();
        System.out.println("Converted to Object Array: " + Arrays.toString(conv));
    }

    public static void LinkedListThing() {
        // Integer Array
        LinkedList<Integer> arr = new LinkedList<Integer>();
        LinkedList<Integer> arr2 = new LinkedList<Integer>();

        // Add item
        arr.add(9);
        arr2.add(12);
        arr2.add(123124);

        // Length
```

```
        System.out.println("Array size: " + arr.size() + ", " + arr2.size());

        // Print
        System.out.println("Print: " + arr.toString() + ", " + arr2.toString());

        // Set
        arr.set(0, 10101010);
        System.out.println("Set 0th index to 10101010: " + arr.toString());

        // Add all
        arr.addAll(arr2);
        System.out.println("Add from arr2 to arr: " + arr.toString() + ", " + arr2.
↪toString());

        // Add to first pos
        arr.addFirst(123);
        System.out.println("Add to first index: " + arr.toString());

        // Add to end pos
        arr.addLast(123);
        System.out.println("Add to last index: " + arr.toString());

        // Clear
        arr2.clear();
        System.out.println("Clear: " + arr.toString() + ", " + arr2.toString());

        // Shallow copy
        System.out.println("Shallow copy: " + arr.clone().toString());

        // Contains
        System.out.println("Contains 1: " + arr.contains(1));

        // Get
        System.out.println("Get 0: " + arr.get(0));
        System.out.println("Get First: " + arr.getFirst());
        System.out.println("Get Last: " + arr.getLast());

        // Find by index
        System.out.println("Return index of object: " + arr.indexOf(123));
        System.out.println("Return last index of object: " + arr.indexOf(123));

        // Offer item to list
        arr.offer(123); // to end
        System.out.println("Offer to end: " + arr.indexOf(123));
        arr.offerFirst(213213);
        System.out.println("Offer to start: " + arr.indexOf(123));
        arr.offerLast(0);
        System.out.println("Offer to end: " + arr.indexOf(123));

        // Peek
        System.out.println("Peek end: " + arr.peek());
        System.out.println("Peek end: " + arr.peekLast());
        System.out.println("Peek first: " + arr.peekFirst());
```

```
        // Poll (find and remove)
        System.out.println("Poll end: " + arr.poll());
        System.out.println("Poll end: " + arr.pollLast());
        System.out.println("Poll first: " + arr.pollFirst());

        // Pop
        System.out.println("Popped: " + arr.pop());

        // Push
        arr.push(1234123);
        System.out.println("Pushed: " + arr.toString());

        // Remove
        arr.remove();
        System.out.println("Remove from head: " + arr.toString());
        arr.remove(0);
        System.out.println("Remove by index: " + arr.toString());
        arr.remove((Integer) 123);
        System.out.println("Remove by object: " + arr.toString());
        arr.removeAll(arr2);
        System.out.println("Remove all items that are contained in other sequence: " +␣
↪arr.toString());
        arr.removeFirst();
        System.out.println("Remove first: " + arr.toString());
        arr.add(123232);
        arr.add(124312312);
        System.out.println("Added: " + arr.toString());
        arr.removeFirstOccurrence(123);
        System.out.println("Remove first occurrence of 123: " + arr.toString());

        arr.add(1234);
        arr.removeLast();
        System.out.println("Remove last: " + arr.toString());
        arr.removeLastOccurrence(123232);
        System.out.println("Remove last occurrence of 123232: " + arr.toString());

        // Convert to array
        ArrayList<Integer> new_ = new ArrayList<Integer>();
        new_.add(123);
        new_.add(123);
        new_.add(21312);
        Object[] conv = new_.toArray();
        System.out.println("Converted to Object Array: " + Arrays.toString(conv));
    }
}


ArrayThings arr = new ArrayThings();
arr.main(new String[1]);
```

```
Integer Array: [1, 2, 3, 4]
Deep Integer Array:[[1, 2], [3, 4]]
Length of array: 4
```

```
Index 0 of intArray: 1
Set index 0 of intArray: [99999, 2, 3, 4]
Swapped: [2, 99999, 3, 4]
Equal Arrays (Shallow equality): false
Equal Arrays (Deep equality): true
BSearch (found): -2
BSearch (not found): -1
Compare (similar): 0
Compare unsigned (similar): 0
Compare (different): 1
Compare unsigned (different): 1
Copy Entire Length: [2, 99999, 3, 4]
Copy Slice: [2, 99999]
Copy Range of Arrays: [2, 99999]
Fill: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Mismatch: 0
Sorted: [2, 3, 4, 99999]
Parallel Sorted: [2, 3, 4, 99999]

Array size: 1, 2
Print: [9], [12, 123124]
Add from arr2 to arr: [9, 12, 123124], [12, 123124]
Clear: [9, 12, 123124], []
Shallow copy: [9, 12, 123124]
Contains 1: false
Get 0: 9
Empty: true
Remove by index: [12, 123124]
Remove by object: [12]
Remove all items that are contained in other sequence: [12]
Keep items that are contained in the other sequence: []
Increased max size: []
Trimmed to size: []
Converted to Object Array: [123, 123, 21312]

Array size: 1, 2
Print: [9], [12, 123124]
Set 0th index to 10101010: [10101010]
Add from arr2 to arr: [10101010, 12, 123124], [12, 123124]
Add to first index: [123, 10101010, 12, 123124]
Add to last index: [123, 10101010, 12, 123124, 123]
Clear: [123, 10101010, 12, 123124, 123], []
Shallow copy: [123, 10101010, 12, 123124, 123]
Contains 1: false
Get 0: 123
Get First: 123
Get Last: 123
Return index of object: 0
Return last index of object: 0
Offer to end: 0
Offer to start: 1
Offer to end: 1
Peek end: 213213
Peek end: 0
```

```
Peek first: 213213
Poll end: 213213
Poll end: 0
Poll first: 123
Popped: 10101010
Pushed: [1234123, 12, 123124, 123, 123]
Remove from head: [12, 123124, 123, 123]
Remove by index: [123124, 123, 123]
Remove by object: [123124, 123]
Remove all items that are contained in other sequence: [123124, 123]
Remove first: [123]
Added: [123, 123232, 124312312]
Remove first occurrence of 123: [123232, 124312312]
Remove last: [123232, 124312312]
Remove last occurrence of 123232: [124312312]
Converted to Object Array: [123, 123, 21312]
```

[ ]: