Chapter 2: Type Conversion

Section 2.1: Numeric primitive casting

Numeric primitives can be cast in two ways. *Implicit* casting happens when the source type has smaller range than the target type.

```
//Implicit casting
byte byteVar = 42;
short shortVar = byteVar;
int intVar = shortVar;
long longVar = intvar;
float floatVar = longVar;
double doubleVar = floatVar;
```

Explicit casting has to be done when the source type has larger range than the target type.

```
//Explicit casting
double doubleVar = 42.0d;
float floatVar = (float) doubleVar;
long longVar = (long) floatVar;
int intVar = (int) longVar;
short shortVar = (short) intVar;
byte byteVar = (byte) shortVar;
```

When casting floating point primitives (`float`, `double`) to whole number primitives, the number is **rounded down**.

Section 2.2: Basic Numeric Promotion

```
static void testNumericPromotion() {

    char char1 = 1, char2 = 2;
    short short1 = 1, short2 = 2;
    int int1 = 1, int2 = 2;
    float float1 = 1.0f, float2 = 2.0f;

    // char1 = char1 + char2;      // Error: Cannot convert from int to char;
    // short1 = short1 + short2;   // Error: Cannot convert from int to short;
    int1 = char1 + char2;          // char is promoted to int.
    int1 = short1 + short2;        // short is promoted to int.
    int1 = char1 + short2;         // both char and short promoted to int.
    float1 = short1 + float2;      // short is promoted to float.
    int1 = int1 + int2;            // int is unchanged.
}
```

Section 2.3: Non-numeric primitive casting

The `boolean` type cannot be cast to/from any other primitive type.

A `char` can be cast to/from any numeric type by using the code-point mappings specified by Unicode. A `char` is represented in memory as an unsigned 16-bit integer value (2 bytes), so casting to `byte` (1 byte) will drop 8 of those bits (this is safe for ASCII characters). The utility methods of the `Character` class use `int` (4 bytes) to transfer to/from code-point values, but a `short` (2 bytes) would also suffice for storing a Unicode code-point.

```
int badInt   = (int)  true; // Compiler error: incompatible types
```

```
char char1   = (char)   65; // A
byte byte1   = (byte)  'A'; // 65
short short1 = (short) 'A'; // 65
int int1     = (int)   'A'; // 65

char char2   = (char) 8253; // ‽
byte byte2   = (byte)  '‽'; // 61 (truncated code-point into the ASCII range)
short short2 = (short) '‽'; // 8253
int int2     = (int)   '‽'; // 8253
```

Section 2.4: Object casting

As with primitives, objects can be cast both explicitly and implicitly.

Implicit casting happens when the source type extends or implements the target type (casting to a superclass or interface).

Explicit casting has to be done when the source type is extended or implemented by the target type (casting to a subtype). This can produce a runtime exception (`ClassCastException`) when the object being cast is not of the target type (or the target's subtype).

```
Float floatVar = new Float(42.0f);
Number n = floatVar;                //Implicit (Float implements Number)
Float floatVar2 = (Float) n;        //Explicit
Double doubleVar = (Double) n;      //Throws exception (the object is not Double)
```

Section 2.5: Testing if an object can be cast using instanceof

Java provides the **`instanceof`** operator to test if an object is of a certain type, or a subclass of that type. The program can then choose to cast or not cast that object accordingly.

```
Object obj = Calendar.getInstance();
long time = 0;

if(obj instanceof Calendar)
{
    time = ((Calendar)obj).getTime();
}
if(obj instanceof Date)
{
    time = ((Date)obj).getTime(); // This line will never be reached, obj is not a Date type.
}
```