COMP 250
INTRODUCTION TO COMPUTER SCIENCE

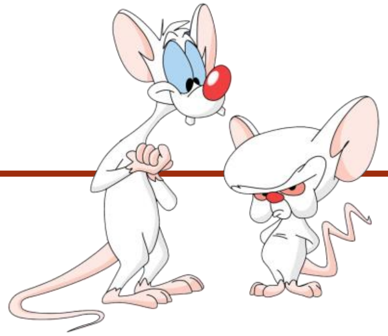
Lecture 4 – Chars and primitive type conversion

Giulia Alberini, Fall 2018

FROM LAST CLASS

- Number Representation
- Binary numbers
- Base conversion
- Binary arithmetic

WHAT ARE WE GOING TO DO TODAY?



- Primitive Data Types
- Char and Unicode
- Type Casting

PRIMITIVE DATA TYPES

PRIMITIVE TYPES

A **primitive** type is

- predefined by the language, and
- named by a reserved keyword

Java supports 8 primitive data types.

THE 8 TYPES SUPPORTED

byte short int long	Integer values
float double	Real Numbers
boolean	True or False
char	One character

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

HOW MANY VALUES?

How many values can you represent with:

- 1 bit?
- 2 bits?
- 3 bits?
- And what about n bits?

$$\boxed{2^n}$$

HOW MANY BITS?

And how many bits do you need to represent:

- 2 different values?
- 4 different values?
- 5 different values?
- And what about x different values?

$$\boxed{\lceil \log_2 x \rceil}$$

➢ So, how many bits do you need to store a boolean?

HOW MANY BITS N DO WE NEED TO REPRESENT A POSITIVE INTEGER M?

$$m = \sum_{i=0}^{N-1} b_i \, 2^i$$

What is the relationship between m and N?

GEOMETRIC SERIES

Recall that,

$$\sum_{i=0}^{N-1} x^i = 1 + x + x^2 + x^3 + \cdots + x^{N-1} = \frac{x^N - 1}{x - 1}$$

That is, if $x = 2$,

$$\sum_{i=0}^{N-1} 2^i = 2^N - 1$$

HOW MANY BITS N DO WE NEED TO REPRESENT A POSITIVE INTEGER M?

$$
\begin{aligned}
m &= \sum_{i=0}^{N-1} b_i \cdot 2^i \\
&\leq \sum_{i=0}^{N-1} 1 \cdot 2^i \\
&= 2^N - 1 \\
&< 2^N
\end{aligned}
$$

Thus,

$$m < 2^N$$

To solve for N, we take the log (base 2) of both sides and obtain the following equation:

$$\boxed{N > \log_2 m}$$

Lower bound

HOW MANY BITS *N* DO WE NEED TO REPRESENT A POSITIVE INTEGER *M*?

Now, let's assume that $N-1$ is the index i of the leftmost bit b_i such that $b_i = 1$.

e.g. We ignore leftmost 0's of the binary representation of m, $(\ldots 00000010011)_2$

Then,

$$m = \sum_{i=0}^{N-1} b_i 2^i = 1 \cdot 2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i \geq 2^{N-1}$$

Taking the log (base 2) of both sides,

$$\log_2 m \geq N-1 \quad \Rightarrow \quad \boxed{N \leq (\log_2 m) + 1}$$

Upper Bound

HOW MANY BITS N DO WE NEED TO REPRESENT A POSITIVE INTEGER M?

We proved that,

$$\log_2 m < N \leq (\log_2 m) + 1$$

Thus, N must be equal to the largest integer less than or equal to $(\log_2 m) + 1$.
We write,

$$N = floor\big((\log_2 m) + 1\big) = \lfloor(\log_2 m) + 1\rfloor$$

where $floor$ means "round down to the nearest integer".

WHY DIFFERENT TYPES?

It turns out that the difference between the types storing integer values and real numbers is the number of bits reserved for those values. For more info: COMP 273

Type	Keyword	Size	Values
Very Small Integer	`byte`	8-bits	$[-128, 127]$
Small Integer	`short`	16-bits	$[-2^{15}, 2^{15}-1]$
Integer	`int`	32-bits	$[-2^{31}, 2^{31}-1]$
Big Integer	`long`	64-bits	$[-2^{63}, 2^{63}-1]$
Low Precision Reals	`float`	32-bits	-
High Precision Reals	`double`	64-bits	-
True/False	`boolean`	1-bit	[true, false]
One character	`char`	16-bits	-

CHAR and UNICODE

CHAR DATA TYPE

We have seen `char` as one of the primitive data types that we have in Java.

Recall:

- we can declare and initialize a variable of type `char` as follows:

```
char letter = ‘a’;
```

- Character literals appears in <u>single quotes</u>
- Character literals can only contain a single character
  - Note that escape sequences are legal because they represent a single character.

UNICODE

- A character set is an ordered list of character, where each character corresponds to a unique number.
- **Unicode** is an international character set. Java uses Unicode to represent characters.
- Variables of type char have 16 bits reserved in the memory to store a value.
- Each character is represented by an integer.
  Note: not every integer represent a character!

Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	Delete

ASCII VS UNICODE

- ASCII: 7 bits. → It can represent 128 characters.
- UNICODE: 16 bits → 65536 characters.
  - It is a superset of ASCII: the numbers 0-127 map to the same characters both in ASCII and Unicode.

CHARACTER ARITHMETIC

- Since every character is practically an integer, we can perform arithmetic operations on variables of type char.

```
char first = 'a';
char second = (char) (first + 1);
```

- What is the value of `second`?
  - 'b'
- Note the typecasting!
  `first` is automatically converted into an integer, and `first + 1` evaluates to 98.
  Then the typecasting converts the `int` into a `char`, and stores 'b' in `second`.

97	61	141	a	a
98	62	142	b	b
99	63	143	c	c
100	64	144	d	d
101	65	145	e	e
102	66	146	f	f
103	67	147	g	g
104	68	150	h	h
105	69	151	i	i
106	6A	152	j	j
107	6B	153	k	k
108	6C	154	l	l
109	6D	155	m	m
110	6E	156	n	n
111	6F	157	o	o
112	70	160	p	p
113	71	161	q	q
114	72	162	r	r
115	73	163	s	s
116	74	164	t	t
117	75	165	u	u
118	76	166	v	v
119	77	167	w	w
120	78	170	x	x
121	79	171	y	y
122	7A	172	z	z

COMPARING CHARS

```
char letter = ‘g’;
if(letter == ‘a’) {
   System.out.println(“first letter of the alphabet”);
} else if (letter == ‘z’) {
   System.out.println(“Last letter of the alphabet”);
} else if (letter > ‘a’ && letter < ‘z’) {
   System.out.println(“Another letter of the alphabet”);
} else {
   System.out.println(“Not a lower case letter of the alphabet”);
}
```

What prints?

- Another letter of the alphabet

TRY IT! - `charRightShift`

Write a method called `charRightShift` which takes a character and an integer `n` as inputs, and returns a character. If the character received as input is a lower case letter of the English alphabet, the method returns the letter of the alphabet which is `n` positions to the right on the alphabet. If the character received as input is not a lower case letter of the English alphabet, then the method returns the character itself with no modifications.

For example:

- charRightShift(‘g’, 2) returns ‘i’,
- charRightShift(‘#’, 2) returns ‘#’

TYPE CASTING

TYPECASTING

- We can convert back and forth between variables of different types using **typecasting**. (or casting, for short)

```
int x = 3;
double y = 4.56;
int n = (int) y;
double m = (double) x;
```

- What are the values of `x, y, n`, and `m`?
  - `x = 3, y = 4.56, n = 4, m = 3.0`

PRIMITIVE TYPE CONVERSION – INT ↔ DOUBLE

- When going from `int` to `double`, an explicit cast is NOT necessary.
- When going from `double` to `int`, you will get a compile-time error if you don’t have an explicit cast.

PRIMITIVE TYPE CONVERSION – IN GENERAL

	number of bits
double	64
float	32
long	64
int	32
char	16
short	16
byte	8

wider

narrower

Here, wider usually (but not always) means more bytes.

EXAMPLES

```
int i = 3;
double d = 4.2;
d = i; // widening (implicit casting)
```

EXAMPLES

```
int i = 3;
double d = 4.2;
d = i; // widening (implicit casting)

d = 5.3 * i; // widening(by "promotion")
```

EXAMPLES

```
int i = 3;
double d = 4.2;
d = i;  // widening (implicit casting)

d = 5.3 * i;  // widening(by "promotion")

i = (int)d;  // narrowing(by casting)
float f = (float) d;  // narrowing  (by casting)
```

EXAMPLES

```
int i = 3;
double d = 4.2;
d = i;  // widening (implicit casting)

d = 5.3 * i;  // widening(by "promotion")

i = (int)d;  // narrowing(by casting)
float f = (float) d;  // narrowing   (by casting)
```

- For primitive types, both widening and narrowing change the bit representation. (See COMP 273.)
- For narrowing conversions, you get a compiler error if you don't cast.

EXAMPLES WITH CHAR

```
char c = 'q';
int x = c // widening
```

EXAMPLES WITH CHAR

```
char c = 'q';
int x = c // widening

c = (char) x; // narrowing
```

EXAMPLES WITH CHAR

```
char c = 'q';
int x = c // widening

c = (char) x; // narrowing

short y = 12;        might negative
c = y; // compile time error!! (need explicit casting)
y = c; // compile time error!! Narrowing → need explicit casting
```

- Next week
  - Monday: packages and modifiers
  - Wednesday: Inheritance
  - Friday: Object class and type conversion