Types

CS110: Introduction to Computer Science

Types

- So far we've worked with three types: `int`, `double`, and `String`
- In this unit, we'll cover some additional types and discuss how the computer stores them.

Additional Built In Types

- Here are the other built-in types we'll care about
  - Integer Types: `byte`, `short`, `int`, and `long`.
  - Floating Point types: `float` and `double`.
  - Letter Types: `char` and `String`.
  - Other: `boolean`.

Integer Types

- Ranges of types:
    - `byte`: -128 to 127 (-2^7 to 2^7-1)
    - `short`: -32,768 to 32,767 (-2^{15} to $2^{15}-1$)
    - `int`: -2,147,483,648 to 2,147,483,647 (-2^{31} to $2^{31}-1$)
    - `long`:-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (-2^{63} to $2^{63}-1$)

Why different sizes?

- Each type of integer takes up a different amount of memory.
- The larger the range the more memory.
- But how are they stored?
- Using Binary!

Binary

- In Binary, aka Base 2, is a system where there are two digits 0 and 1.
- We can make ANY number using these two digits.
- To see how, let's look at Base 10, aka, decimal, aka, what you're use to.

Base-10

$$6{,}349$$

$$\frac{6}{1{,}000} \quad \frac{3}{100} \quad \frac{4}{10} \quad \frac{9}{1}$$

$$\frac{6}{10^3} \quad \frac{3}{10^2} \quad \frac{4}{10^1} \quad \frac{9}{10^0}$$

$$6 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 9 \times 10^0$$

Base 2 (Only 0 & 1)

Each 0 or 1 value is called a bit in binary (like a digit in decimal)

$$0110$$

$$\frac{0}{2^3} \quad \frac{1}{2^2} \quad \frac{1}{2^1} \quad \frac{0}{2^0}$$

$$\frac{0}{8} \quad \frac{1}{4} \quad \frac{1}{2} \quad \frac{0}{1}$$

$$0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

Converting Base 10 to 2

1. Find largest Base 2 number that is smaller than your base 10.
2. Subtract that value, and put a 1 in the associated base 2 bit.
3. Take new value, and replay steps 1 & 2 until you have 0.
   - **Let's do a few examples.**

Byte

- It's often convenient to group bits into something more manageable.
- 8 bits is a **byte**.
- This is the basic until of our system
- We have 8-bit systems (1 byte), 16-bit (2 bytes), 32 bit (4 bytes), and 64 bit (8 bytes).
- In one byte, the largest number you can fit is 2^8-1 or 255.

Space of types

- Ranges of types:
    - `byte`: 8 bit
    - `short`: 16 bit
    - `int`: 32 bit
    - `long`: 64 bit

Space of types

- Ranges of types:
  - `byte`: 8 bit
  - `short`: 16 bit
  - `int`: 32 bit
  - `long`: 64 bit
- **You might wonder,** If byte is 8 bits, why is the largest number 2^8? Why not 2^8-1

Negative Numbers

- We're not going to go exactly into how to computers store negative numbers
  - We cover it in CS 221
- The general idea though, is that the first bit is the "sign".
  - If it's 0, the number is positive and if its 1 its negative
- Also, this means "0000 0000" counts as a "positive" number, which is why there is one more negative number than positive.
  - e.g., -128 to 127 (-2^7 to 2^7-1)

Floating Point

- There are two types of decimal number `float` and `double`.
- `float` is 32 bit
- `double` is 64 bit.
- These are called **floating point** numbers because they can trade off precision for range.
  - i.e., Smaller numbers can be stored with more precision than larger numbers
- As a result, max & min values don't mean as much for these too

How do they store data

- Not going to cover this in detail, but there is one important point.
- Floating points types use bits like integers.
- However, they allow for bits to be placed in 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4},...
  - e.g. 0.5, 0.25, 0.125, 0.0625

Why this matters.

- Because some "finite" numbers in base 10 cannot be represented exactly in base 2
  - e.g., 0.2 in base 10 is .001100110011... (repeating) in base 2
- Also, irrational numbers and even base-10 repeating number cannot be represented exactly in base 2.

This matters

- This matters because when you compare **floating point numbers you must take this imprecision into account.**

```
double x = 11111111111111111111.0/55555555555555555555.0;
double y = 2.0/10.0;
if(x==y){
    Magic.println("yay");
} else {
    Magic.println(x+" doesn't equal "+ y);
}
```

Two approaches

- How to fix this
  1. **Don't care**. Sometimes, it doesn't really matter. So, if you don't need an exact answer you can ignore it
  2. **Compare the absolute difference**. You can use **`Math.abs()`** to compare the difference between two doubles. You pick the tolerance (aka, epsilon)

Characters

- `char` represents one character.
- Like, everything else it is represented using bits
- Characters use to be one byte **ASCII** characters
- Let's look at the chart on the next slide

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Characters

- Currently, Java represents numbers using **unicode**.
- It's big enough to be able to represent every character in every language
- https://www.ssec.wisc.edu/~tomw/java/unicode.html
- Let's look at Basic Latin.

Strings

- `String`s are unique among the built-in type in that they are built using another type as their base.
  - i.e., `String`s are arrays of `char`.
- All of the other built in types are called **primitives**.

Boolean

- `boolean` is a one bit value that is `true` or `false`,

Modifier

- We can modify any of the primitive types by adding the keyword `final`.
- This makes it a constant and means we cannot modify it after assigning it a value.

Making New Types

Misspelling

- What happens when you misspell "cyan" when drawing?
- Wouldn't it be nice if we could fix this?
- Yes, we can making a new type that has spell check "built it"

Enumeration

- Enumerations (**`enum`**) is how we can create a type that has a limited number of human readable labels
- Syntax is simple

```
public enum CSColor{
    Red,Blue, Green, Cyan
}
```

- You can then, use it like any other type.
- **Let's try an example**

Where to Declare `enum`

- Enums can be in their own files.
- Enums can live in a `class` file, within in the `{}`
  - If it's declared in a class file, then is scoped within the class

Enums and strings

```java
public enum Work {
    school, home, office, superBoring
}
```

- Two methods allow us to go back and forth between strings and enums: `toString()` and `valueOf()`.
- In the example, notice that `toString()` is called on a variable, while `valueOf()` is declared on the enum itself.

```java
Work my = Work.home;
System.out.println(my.toString());

Work your = Work.valueOf("superBoring");
System.out.println(your.toString());
```

All Values

```java
public enum Work {
    school, home, office, superBoring
}
```

- The methods `values()` will get you an array of all possible values in an enum.

```java
Work[] test = Work.values();
for(int i =0;i<test.length;i++) {
  System.out.println(test[i].toString());
}
```

All values

`switch` statement

- The `switch` statement is an improved version of if-then-else designed for enums.

```
CSColor c = CSColor.Red;
switch(c){
    case Red:
        Magic.println("It's red");
        break;
    case Cyan:
        Magic.println("It's cyan");
        break;
    default:
        Magic.println("It's something else");
}
```

- **Let's try this and the `if` equivalent.**

`break`

- By default, `switch` statements **fall-through**
  - **Let's see this in action**
- `break` statements stop the **fall-through**

`default`

- `default` is the "`else`" statement of `switch`.
  - Let's see this in action.

Spreadsheets

Spreadsheets

- OK, we can grow now lets switch gears.
- How do make a spreadsheet.

	A	B	C	D
1	100	200	-50	65
2	73	22	66	88
3	123	127	-2	9

- What is this?
- Well, a list of lists

Two Dimensional Array

```
int[][] weird = new int[2][3];
weird[0][0] = 20;
weird[0][1] = 40;
weird[0][2] = 50;
weird[1][0] = 60;
weird[1][1] = 70;
weird[1][2] = 80;
```

For Loops & 2D Array

```
int sum = 0;
for(int row = 0;row< weird.length;row++){
    for(int col = 0;col<weird[row].length;col++){
        weird[row][col] += sum;
    }
}
System.out.println(sum);
```

```java
int sum = 0;
for(int row = 0;row< weird.length;row++){
    for(int col = 0;col<weird[row].length;col++){
        weird[row][col] += sum;
    }
}
System.out.println(sum);
```

```java
int sum = 0;
for(int[] weirdRow: weird){
    for(int element: weirdRow){
        element += sum;
    }
}
System.out.println(sum);
```

Multidimensional

- Can we do a 3D? 4D? 10D? YES!

```
int [][][] threeD = new int[5][2][7];
int [][][][] fourD = new int[5][10][2][7];
int [][][][][][][][][][] tenD = new int[5][10][2][7][1][6][3][20][2][3];
```

- Let's talk about why you might want to do this.