Today

- Manipulating primitive data types
- Eclipse & submitting projects

Short Questions?

Quiz

Quiz grading:

- Most questions: equal grade weight is given to participation and correctness
- Some questions are graded by participation only
- Procedure update: individual quizzes will be added to the D2L grade book on an individual basis
  - The 15% of final grade will be distributed evenly across all quizzes that we take
  - Drop the lowest two

Example: “=“ operator

```
int foo;
foo = 5;
foo = foo + 3;
System.out.println(foo);
```

”=“ is about storage, not equality!

Juggling Exercise

Handing In a Project

Process:

- Write, test and debug the code
- Export project to a Zip file
- Submit to D2L dropbox

Exporting a Project

- Select the project in the Package Explorer
- File: Export
- Export destination: General: Double click on “Archive File”
- To archive file: Give the name of the zip file
  - Note: you may also have to browse to a destination folder
- Leave “Save in zip format” selected
- Click Finish

Mathematical Operators

- Satisfy standard precedence relationships:
  - Level 2: ++ --
  - Level 3: ( ) for grouping of expressions
  - Level 4: * / %
  - Level 5: + -
- Each operator is potentially defined differently for different data types

Some Syntactic Notes

Curly brackets {} and parentheses () always come in matching pairs

- {}: used to group several statements together
- (): used for method (or function) definition/calls
- Eclipse helps you to keep track of these pairs by:
  - Indenting code within {}
  - Giving errors when one of a pair is missing

Semicolons (;) are necessary to end a single code statement.

- Eclipse will also give you an error if you have forgotten one

Camel Case Convention

- We try to make our identifiers as descriptive as possible by describing them with multiple words
- However, a space character cannot be used as part of an identifier
- So, we cram the words together:

```
int numberOfCamels;
```

- Note:
  - First letter of a variable name is (by convention) lower case
  - But the first letter of a class name is upper case

final keyword

- Some of our variables are not actually variable – they are constants
- One could just include the value in the code, but these “magic values” are not very descriptive & make the code hard to read and maintain
- Instead, we want to use a descriptive identifier, but we want the compiler to enforce the fact that it will not change
- Convention is that these constants use only capital letters

```
final double CM_PER_INCH = 2.54;
```

Characters

- An individual character is stored in a single byte
- Since a byte is just a number, we must have some way of mapping numbers to glyphs (the visual representation of a character)

ASCII Encoding of Characters

- This encoding served us well for many years
- But, we really want to be able to represent any glyph
- Answer: Unicode uses multiple bytes to capture a single character (the number of bytes depends on the standard)

Binary	Dec	Hex	Glyph
010 0000	32	20	SP
010 0001	33	21	!
010 0010	34	22	"
010 0011	35	23	#
010 0100	36	24	$
010 0101	37	25	%
010 0110	38	26	&
010 0111	39	27	'
010 1000	40	28	(
010 1001	41	29	)
010 1010	42	2A	*
010 1011	43	2B	+
010 1100	44	2C	,
010 1101	45	2D	-
010 1110	46	2E	.
010 1111	47	2F	/
011 0000	48	30	0
011 0001	49	31	1
011 0010	50	32	2
011 0011	51	33	3
011 0100	52	34	4
011 0101	53	35	5
011 0110	54	36	6
011 0111	55	37	7
011 1000	56	38	8
011 1001	57	39	9
011 1010	58	3A	:
011 1011	59	3B	;
011 1100	60	3C	<
011 1101	61	3D	=
011 1110	62	3E	>
011 1111	63	3F	?

Binary	Dec	Hex	Glyph
100 0000	64	40	@
100 0001	65	41	A
100 0010	66	42	B
100 0011	67	43	C
100 0100	68	44	D
100 0101	69	45	E
100 0110	70	46	F
100 0111	71	47	G
100 1000	72	48	H
100 1001	73	49	I
100 1010	74	4A	J
100 1011	75	4B	K
100 1100	76	4C	L
100 1101	77	4D	M
100 1110	78	4E	N
100 1111	79	4F	O
101 0000	80	50	P
101 0001	81	51	Q
101 0010	82	52	R
101 0011	83	53	S
101 0100	84	54	T
101 0101	85	55	U
101 0110	86	56	V
101 0111	87	57	W
101 1000	88	58	X
101 1001	89	59	Y
101 1010	90	5A	Z
101 1011	91	5B	[
101 1100	92	5C	\
101 1101	93	5D	]
101 1110	94	5E	^
101 1111	95	5F	_

Binary	Dec	Hex	Glyph
110 0000	96	60	`
110 0001	97	61	a
110 0010	98	62	b
110 0011	99	63	c
110 0100	100	64	d
110 0101	101	65	e
110 0110	102	66	f
110 0111	103	67	g
110 1000	104	68	h
110 1001	105	69	i
110 1010	106	6A	j
110 1011	107	6B	k
110 1100	108	6C	l
110 1101	109	6D	m
110 1110	110	6E	n
110 1111	111	6F	o
111 0000	112	70	p
111 0001	113	71	q
111 0010	114	72	r
111 0011	115	73	s
111 0100	116	74	t
111 0101	117	75	u
111 0110	118	76	v
111 0111	119	77	w
111 1000	120	78	x
111 1001	121	79	y
111 1010	122	7A	z
111 1011	123	7B	{
111 1100	124	7C	|
111 1101	125	7D	}
111 1110	126	7E	~

Special Characters

- ‘\n’ is a single character that means “new line” (and is often implemented as both a “new line” and a “carriage return”)
- ‘\t’ is a tab
- ‘\\’ is a \

Mixing Types with Operators

```
int foo = 4;
double bar = 5.3;
System.out.println(foo + bar);
```

- For the + operation: the value 4 is first converted to a double; then, it is added to 5.3
- The result (a double) is then converted to a string for use by `println()`

Mixing Types with Operators

- Not all conversions are automatic (in fact, few are
  - A double will not be converted automatically to an int
- We tell the Java compiler that such a conversion is allowed through casting:

```
int foo;
double bar = 5.3;

foo = (int) bar;
System.out.println(foo);
```

Casts are rare. If you think you need it, something might be wrong in your design or implementation

Wrap Up

Due this week:

- HW 1: Turing's Craft
- Project 0: Eclipse + D2L

Also out:

- HW 2: due next week

Next time:

- Conditionals