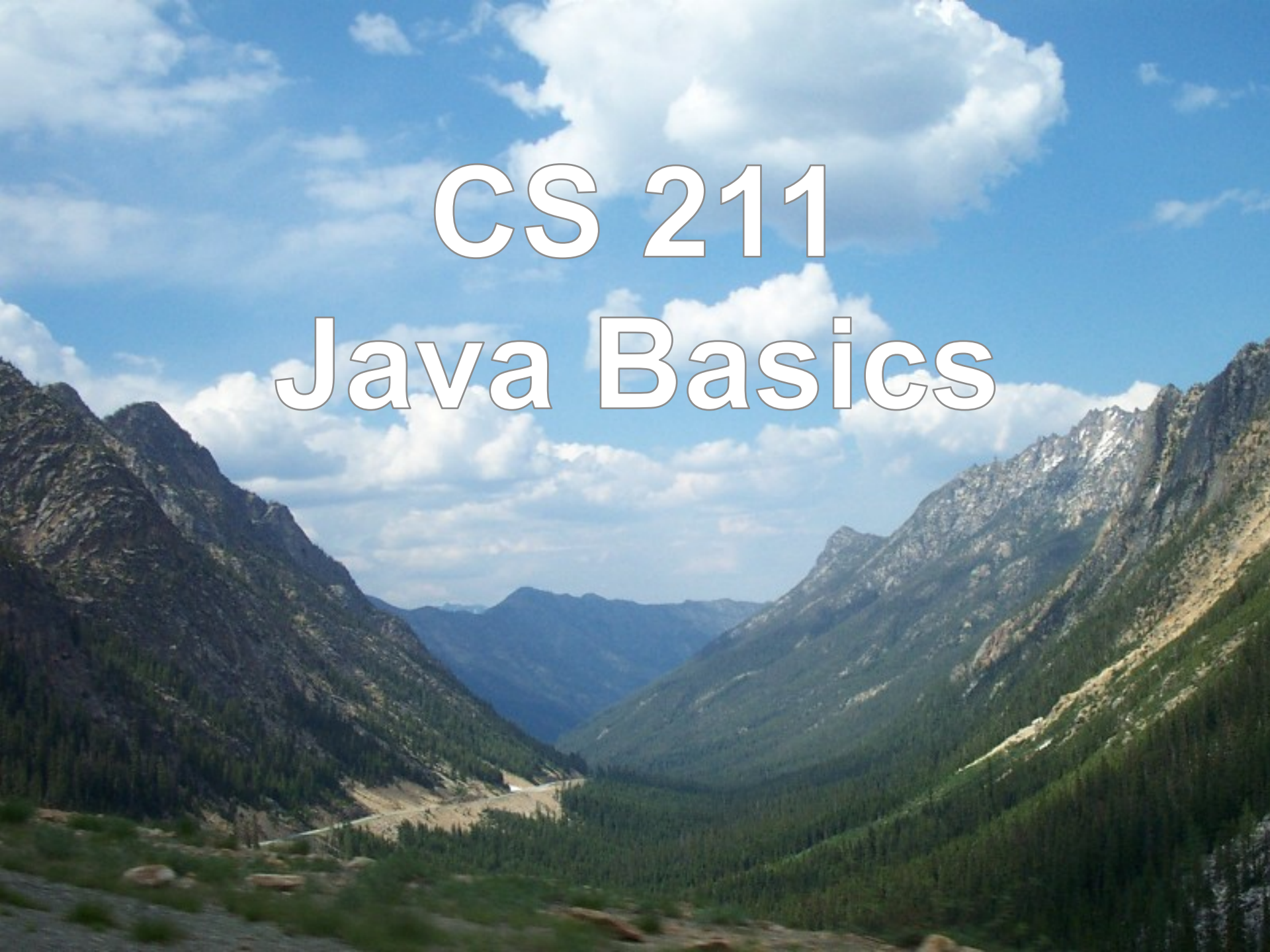
CS 211
Java Basics

Simple Java Program

filename: HelloWorld.java

```
public class HelloWorld {
    public static void main (String[] args) {
        //our instructions go here
        System.out.println("Hello, World!");
    }
}
```

Each word of this should make sense by the semester's end! For now it is boilerplate code—just the template we'll use to write code.

Whitespace

- whitespace includes all 'blank' characters:
  - space, tab, newline characters
  - whitespace is (almost) irrelevant in Java.
  - spaces used to separate identifiers (int x vs intx)
  - we can't span lines within Strings. (no <enter> between quotes)

- Syntax is not based on indentations
  - but indentation is highly recommended! (required for class)

Bad Whitespace Example #1

Valid, but horribly written, code.
(excessive, meaningless spacing)

```
public                          class
    Spacey { public
        static

void
        main(String[
                ] args
            ){ System      .
           out.println("Weirdly-spaced code still runs!"

                    );}}
```

Bad Whitespace Example #2

Valid, but horribly written, code.
(one-liners aren't always best!)

```
public class Spacey2{public static void main(String[]args{System.
out.println("space-devoid code also runs...");}}
```

(the above is all on one line of text in Spacey2.java)

Code like this might not receive any credit! Seriously, don't do this in anything you ever turn in. Never make the grader unhappy.

Good Whitespace Example

```
public class GoodSpacing {
  public static void main (String[] args) {
    int x = 5;
    int y = 12;
    System.out.println("x+y = " + (x+y));
  }
}
```

indentation levels for each block: class, method definitions, control structures…

Identifiers

- Identifiers are the names we choose for variables, methods, classes, interfaces, etc.
  - can use letters, digits, underscore(_), and dollar ($)
  - Identifiers cannot begin with a digit
  - you can't use Java's keywords as identifiers
  - Java is *case sensitive*: `Total`, `total`, `TOTAL` are distinct

convention: identifiers

- programmers choose different styles for different types of identifiers:
  - *lower case* variables: `count, distToEmpty`
  - *title case* classes: `Person, MasonStudent`
  - *upper case* constants: `MASON, MAX_INT`

Identifier Examples

Legal Identifier Examples:

hello

camelCaseName

__$_09abizzare

user_input18

anyArbitrarilyLongName

Illegal Identifier Examples:

two words

Extra-Characters!

1st_char_a_digit

transient *(it's a keyword)*

Dots.And.Hooks?

Java Keywords

Keywords are part of the language definition.
Their only meaning is the original intent -
programmers can't use them as new identifiers.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

greyed-out keywords are ones we won't learn in this course.

Pytania Poll

Java Basics.

Types!

Java is strongly typed

- every expression has a specific type, known at compile time
- an expression's type never changes – whether it's a variable, literal value, method call, or any other expression

Java has two kinds of types:

- **primitive** types (containing literal values)
- **reference** types (containing objects of some class)

Primitive Types

- the basic values of the language:
  numbers, characters, and booleans

boolean: truth values. Only possible values:

```
true  false
```

char: one character in single-quotes. examples:

```
'a'    'H'    '\n'    '5'
```

numbers: many versions of integers, two float types.

- each has a finite range

Integer Types

Each integral type has its own finite range:

byte	(8 bits)	-128	→	127
short	(16 bits)	-32768	→	32767
int	(32 bits)	-2147483648	→	2147483647
long	(64 bits)	(-2^{63})	→	$(2^{63}-1)$

---

char (16 bits) 0 → 65535 (all positive)

(can edit char as its Unicode #, but it still is printed as its current character, not the code number)

- The compiler can't use out-of-range numbers.
- long constants need an 'L' suffix (or lowercase 'l'):

```
123412341234L      037L      0x345L      -100000L
```

Integer Representation

- ***decimal***: no leading zeroes; plain base-ten.
  0 10 483 –9876501234 66045

- ***hexadecimal*** (base 16): prefix 0x, followed by one or more of 0123456789ABCDEF. (a-f are equivalent to A-F).
  0x0 0xfade 0x1B2C 0x9 –0x10

- ***octal*** (base 8): prefix **0**, followed by one or more of 0-7.
  00 071 –045306 01777 010

Note on Different Representations

All three inputs are alternatives you can use to describe the **same** values.

You also know:

Roman Numerals	(e.g., XLVI)
tally-marks 𝍸	(base 1 counting)

→ All of these represent integers! Don't confuse representation with meaning.

Floating Point Numbers

Approximating the real #s: called floating point numbers.

We just write things in normal base 10 as always.

internal binary representation: like scientific notation.

S: sign bit. E: bias-adjusted exponent.

M: adjusted fractional value.

value = $(-1)^S * 2^E * M$

Also representable: infinity (+/−), NaN ("not a number")

float: 32-bit representation. (1 sign, 8 exp, 23 frac)

double: 64-bit representation. (1 sign, 11 exp, 52 frac)

Representing Floating Point Numbers

Floating Point numbers may be:

- a decimal point followed by digits `2.32` `1.21` `450000`
- written in scientific notation: `2.32e5` `6.0221409e23`
- may be very large: `2E35F` `2e250` `-2e250`

float: use `f` suffix (or `F`) to indicate float.

`0f` `3.14159f` `-2E3F` `59023f`

double: floating point numbers are doubles by default.

`0.3` `3.141592653589` `-3.15E30`

Creating Variables

Variables must be **declared** and **initialized** before use.

Declaration: creates the variable. It includes a type and a name. The variable can only hold values of that type.

```
int x;      char c;     boolean ok;          Person p;
```

Initialization: assign an expr. of the variable's type to it.

```
x=7+8;    c='M';          ok = true;        p = new Person();
```

Both: we can declare and instantiate all at once:

```
int x = 5;          char c = 'S';       Person p = new Person();
```

Casting

changing between numerical types is possible,
but has implications.

- a **cast** operation is a conversion from one type to another.
- place the desired type in parentheses in front of the value:

```
(int) 3.14
```

One use: forcing floating-point division.

```
int x=3, y=4;
double z = ((double)x)/y;
System.out.println(z);      //prints 0.75
```

Pytania Poll

Primitive Types.

Java Comments

There are two main styles of comments in Java:

- **Single-Line**: from **//** to the end of the line.

- **Multi-Line**: all content between **/*** and ***/**, whether it spans multiple lines or is within one line.

- **JavaDoc**: a convention of commenting style to auto-generate documentation/API. More on this later. (done by special uses of /* */ comments)

Expressions, Statements

Expression: a representation of a calculation that can be evaluated to result in a single value. There is no indication what to do with the value.

Statement: a command, or instruction, for the computer to perform some action. Statements often contain expressions.

Basic Expressions

- literals (all our numbers, booleans, characters)
- operation exprs:

< <= > >= == !=	(relational ops)
+ - * / %	(math ops)
&& || !	(boolean ops)
e ? e : e	(ternary conditional expr)

- variables
- parenthesized expressions (expr)

Conditional Expression

Ternary Operator ?: boolexpr ? expr : expr

- a conditional *expression*: it evaluates the boolean expression, and then results in the middle expression when true, and the last expression when false.

- We **must have all three parts**
- the 2nd and 3rd expressions' types must agree with (fit in) the resulting type of the entire expression.

Expression Examples

Legal:

4+5
x%2==1
numPeople
(2+3)*4

(3>x) && (! true)
(x<y)&&(y<z)
drawCard()
y!=z

Illegal (these aren't expressions):

x>y>z 4 && false 7(x+y)

Basic Statements

- **Declaration**: announce that a variable exists.
- **Assignment**: store an expression's result into a variable.
- **method invocations** (may be stand-alone)
- **blocks**: multiple statements in **{ }**'s
- **control-flow**: if-else, for, while, … *(next lecture)*

Statement Examples

```
int x;                          // declare x
x = 15;                         // assignment to x
int y = 7;                      // decl./assign. of y
x = y+((3*x)-5);                // assign. with operators
x++;                            // increment stmt (x = x+1)
System.out.println(x);// method invocation


if (x>50) {                     // if-else statement
      x = x - 50;
}
else {
      y = y+1;
}
```