Introduction to Computational Thinking

Lecture #3:

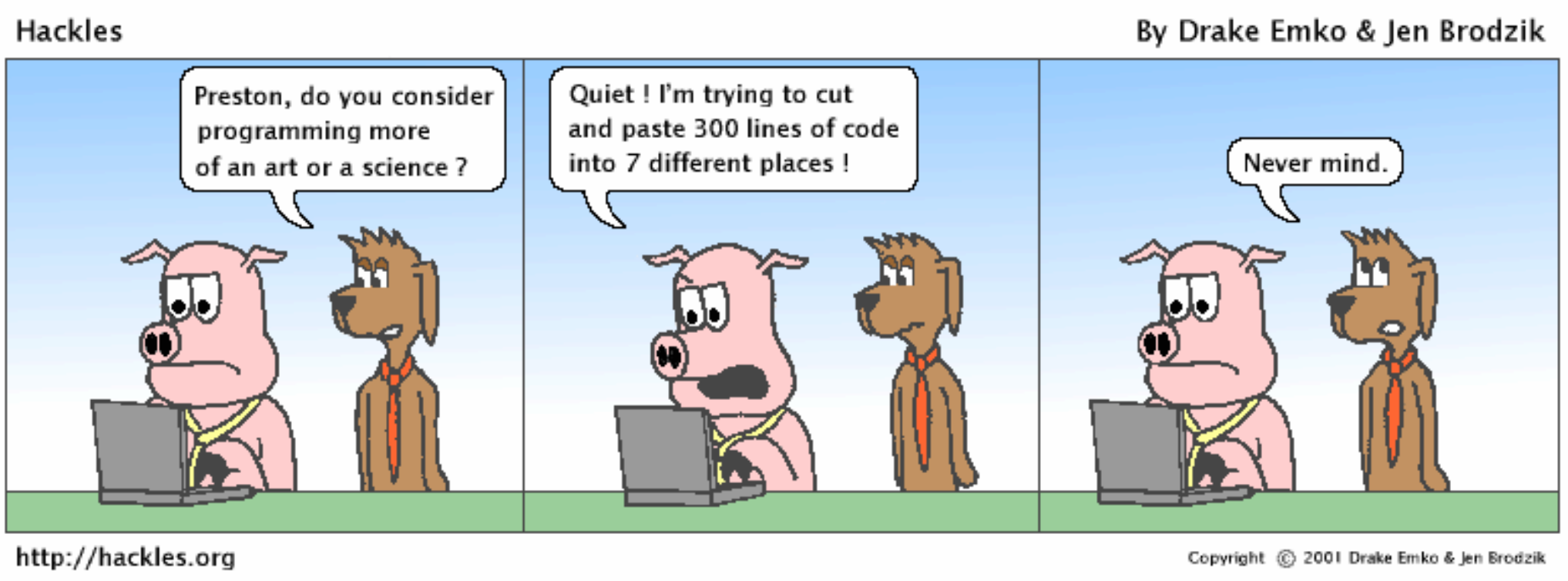
Java Methods & Primitive Data Types

Qiao Xiang, Qingyu Song

https://sngroup.org.cn/courses/ct-xmuf25/index.shtml

10/15/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.


Hackles
By Drake Emko & Jen Brodzik
Preston, do you consider programming more of an art or a science ?
Quiet ! I'm trying to cut and paste 300 lines of code into 7 different places !
Never mind.
http://hackles.org
Copyright © 2001 Drake Emko & Jen Brodzik

Outline

- Admin.
- Java methods

Admin

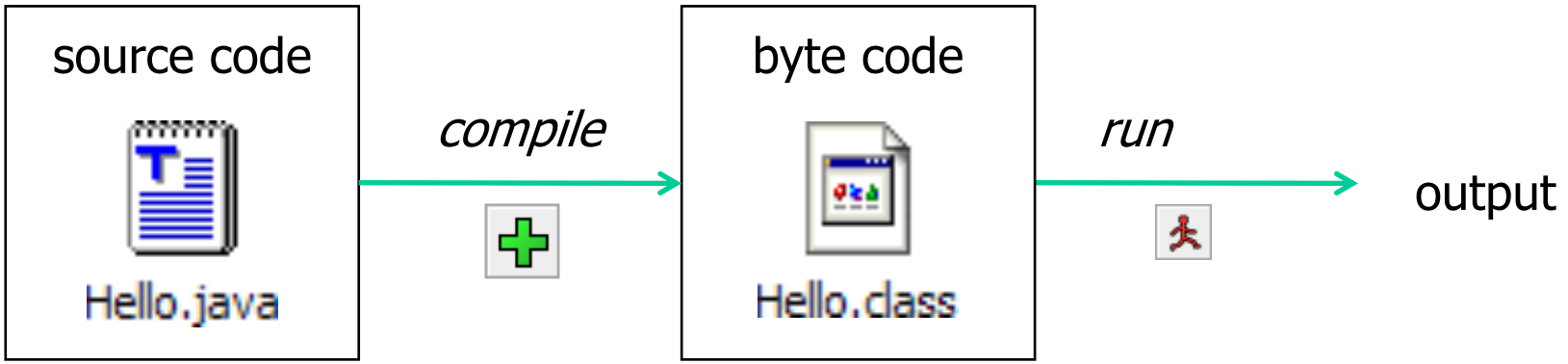
- Practice Slides at the end of slides for Lecture 2
- Office hours have been posted
- PS1
  - You have 9 discretionary late days across the semester, but can use at most 3 days per PSET

Recap

- What are digital symbol manipulations?
- Data are digital symbols
- Programs are digital symbols manipulation

Recap: Java Programming Steps

- Programming in Java consists of three tasks
  - edit java source code (.java files)
  - compile java source code to generate bytecode (.class files)
  - execute/run/test bytecode using an interpreter



Recap: Top-Down Java Syntax Structure

A class:
- has a name, defined in a file with same name
  Convention we follow: capitalize each English word
- starts with {, and ends with }
- includes a group of methods

```
public class <class name> {
    public static void main(String[] args) {

        <statement>;
        <statement>;
        …
        <statement>;

    }
}
```

A method:
- has a name
  Convention we follow: lowercase first word, capital following
- starts with {, and ends with }
- includes a group of statements

statement:
- a command to be executed
- end with ;

Recap: The `System.out.println` Statement

- Two ways to use the statement:
  - `System.out.println("string");`
    - You may need to use escape sequences in strings
  - `System.out.println();`

- A related statement is
  `System.out.print("string");`
  It does not print a newline

Java Syntax: A Bottom-Up Look

- Basic Java syntax units
  - white space and comments
  - identifiers (words)
  - symbols: { } " ( ) < > [ ] ; = …
  - strings
  - numbers

```
// This is a one-line comment
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

Java Syntax: A Bottom-Up Look

- Basic Java syntax units
  - white space and comments
  - identifiers (words)
  - symbols: { } " ( ) < > [ ] ; = …
  - strings
  - numbers

Java depends on the identifiers and symbols to understand your program

```
public class Hello {
    [redacted]
}
```

Syntax Error: Example

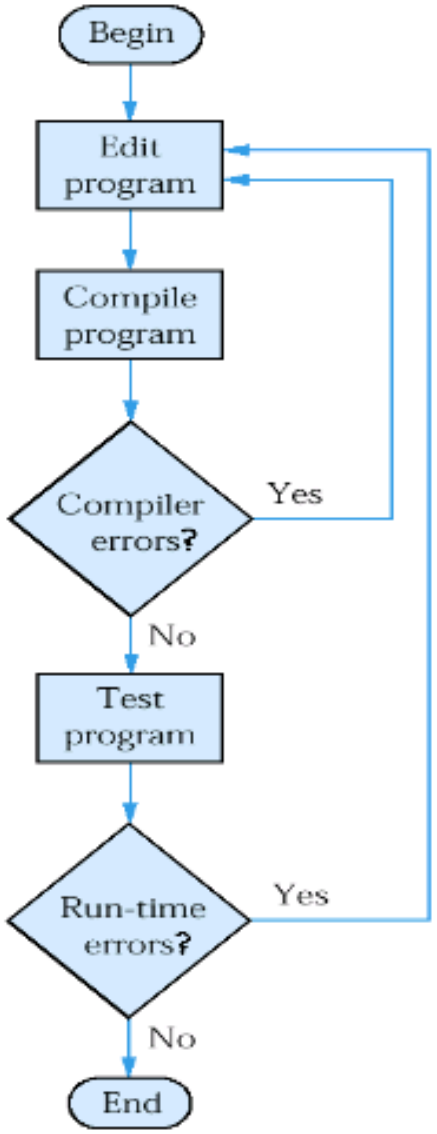
```
1  public class Hello {
2      pooblic static void main(String[] args) {
3          System.owt.println("Hello, world!")_
4      }
5  }
```

- Compiler output:

```
Hello.java:2: <identifier> expected
    pooblic static void main(String[] args) {
          ^
Hello.java:3: ';' expected
}
^
2 errors
```

  - The compiler shows the line number where it found the error.
  - The error messages sometimes can be tough to understand:
    - Why can't the computer just say "*You misspelled 'public'*"?
    - Since the computer knows that a ";" is missing, can't it just fix it??

Java Programming Steps and Errors



- *Compile-time errors*
  - the compiler may find problems with syntax and other basic issues
  - if compile-time errors exist, an executable version of the program is not created
- *Run-time errors*
  - a problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (crash)
- *Logical errors*
  - a program may run, but produce incorrect results

Roadmap

any program you might want to write

objects

methods and classes

graphics, sound, and image I/O

arrays

conditionals and loops

math | text I/O

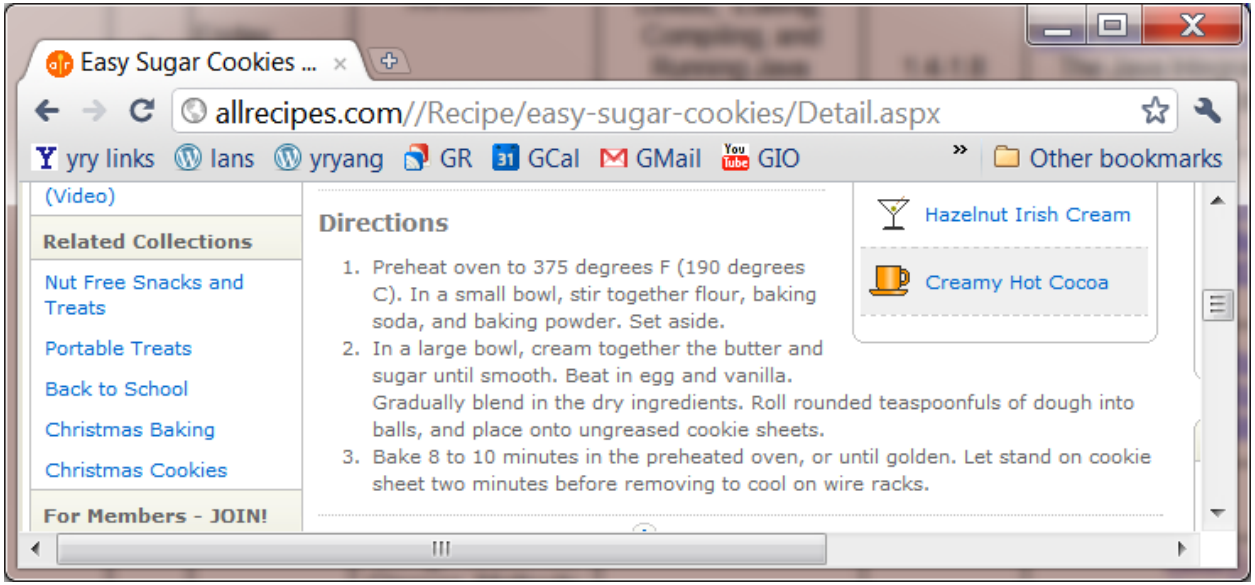
primitive data types | assignment statements

Outline

- Admin.
- von Neumann Model: a Symbol Manipulation Platform
- Java methods
  - Motivation: why methods?

Algorithms

- **Algorithm**: A list of steps for solving a problem.
- An example algorithm (recipe): "Bake sugar cookies"



An Example Algorithm Spec: "Bake two batches of sugar cookies"

1. Preheat oven temperature to 375F.
2. Mix the dry ingredients.
3. Cream the butter and sugar.
4. Beat in the eggs.
5. Stir in the dry ingredients.
6. Set the timer for 8 min.
7. Place 1st batch of cookies to oven.
8. Allow the cookies to bake.
9. Set the timer for 8 min.
10. Place 2nd batch of cookies to oven.
11. Allow the cookies to bake.
12. Mix ingredients for frosting.
13. Spread frosting and sprinkles.

Readability of the specification?

Problem 1: Lack of Structure

- *Lack of structure*: Many tiny steps; tough to remember.
  - A human being typically can only manage seven (plus or minus 2) pieces of information at one time

http://www.michaeljemery.com/nlp/your-conscious-minds-capacity-seven-plus-or-minus-two-chunks-of-information/

Problem 2: Redundancy

- *Redundancy: unnecessary repeat*

1. Preheat oven temperature to 375F.
2. Mix the dry ingredients.
3. Cream the butter and sugar.
4. Beat in the eggs.
5. Stir in the dry ingredients.
6. Set the timer for 8 min.
7. Place the first batch of cookies into the oven.
8. Allow the cookies to bake.
9. Set the timer for 8 min.
10. Place the second batch of cookies into the oven.
11. Allow the cookies to bake.
12. Mix ingredients for frosting.
13. Spread frosting and sprinkles.

Fix: Structured Algorithms

- **Structured algorithm**: Split into coherent tasks.

1 Preheat oven.

- Set oven to 375 degrees

2 Make the cookie batter.

- Mix the dry ingredients.
- Cream the butter and sugar.
- Beat in the eggs.
- Stir in the dry ingredients.

3 Bake the cookies.

- Set the timer for 8 min.
- Place the cookies into the oven.
- Allow the cookies to bake.

4 Decorate the cookies.

- Mix the ingredients for the frosting.
- Spread frosting and sprinkles onto the cookies.

Structured Algorithm?

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies2 {
    public static void main(String[] args) {
        // Step 1: preheat oven
        System.out.println("Preheat oven to 375F.");

        // Step 2: Make the cookie batter.
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");

        // Step 3a: Bake cookies (first batch).
        System.out.println("Set the timer for 8 min.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 3b: Bake cookies (second batch).
        System.out.println("Set the timer for 8 min.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 4: Decorate the cookies.
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```

Structured Algorithms

- **Structured algorithm** provides abstraction (hide/ignore the right details at the right time)

1 Preheat oven.

- Set oven to 375 degrees

2 Make the cookie batter.

- Mix the dry ingredients.
- Cream the butter and sugar.
- Beat in the eggs.
- Stir in the dry ingredients.

3 Bake the cookies.

- Set the timer.
- Place the cookies into the oven.
- Allow the cookies to bake.

4 Decorate the cookies.

- Mix the ingredients for the frosting.
- Spread frosting and sprinkles onto the cookies.

Structured Algorithms

- **Structured algorithm** provides abstraction (hide/ignore the right details at the right time)

1 Preheat oven.

2 Make the cookie batter.

3 Bake the cookies.

4 Decorate the cookies.

Removing Redundancy

- A well-structured algorithm can describe repeated tasks with less redundancy.

1 Preheat oven.

2 Make the cookie batter.

3a Bake the cookies (first batch).

3b Bake the cookies (second batch).

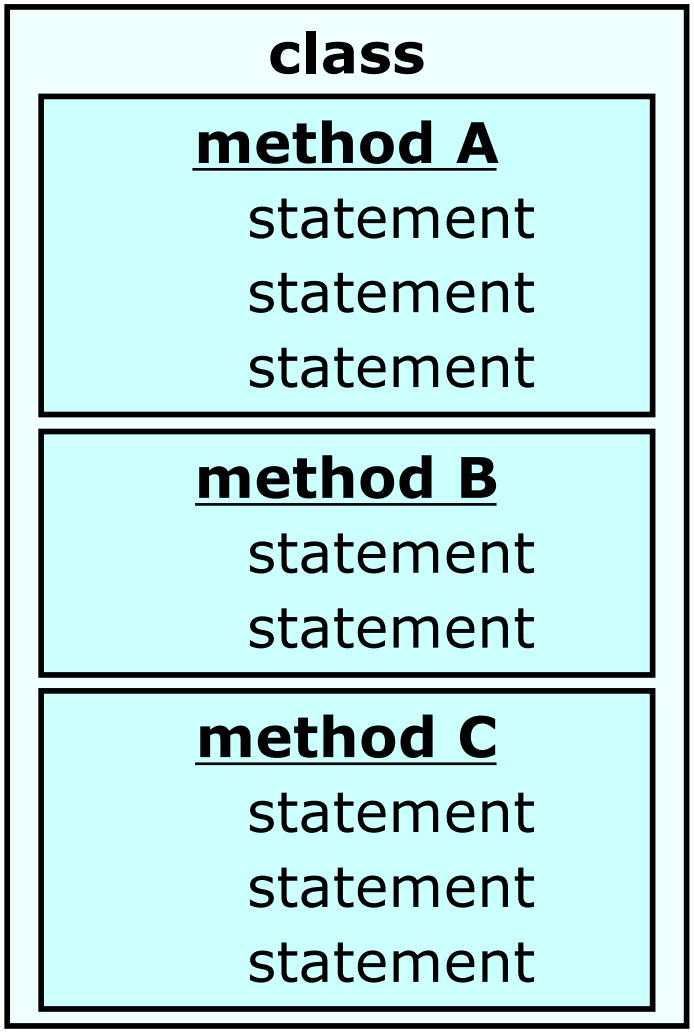
4 Decorate the cookies.

Outline

- Admin.
- Java methods
  - Motivation
  - Syntax: declaring method

Static Methods

- Arrange statements into groups and give each group a name.
- Each such named group of statements is a static method
- Writing a static method is like adding a new command to Java.



Declaring a Method

Gives your method a name so it can be referred to.

- Syntax:

```
public static void <name>() {
    <statement>;
    <statement>;
    …
    <statement>;
}
```

- Example:

```
public static void printWarning() {
    System.out.println("This product causes cancer");
    System.out.println("in lab rats and humans.");
}
```

Calling a Method

Executes the method's code

- Syntax:

```
<name>();
```

  - You can call the same method many times if you like.

- Example:

```
printWarning();
```

  - Output:

```
This product causes cancer
in lab rats and humans.
```

Example

```
public class FreshPrince {
    public static void main(String[] args) {
        rap();                  // Calling (running) the rap method
        System.out.println();
        rap();                  // Calling the rap method again
    }

    // This method prints the lyrics to my favorite song.
    public static void rap() {
        System.out.println("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

Example

```
public class FreshPrince {
    public static void main(String[] args) {
        rap();                  // Calling (running) the rap method
        System.out.println();
        rap();                  // Calling the rap method again
    }

    // This method prints the lyrics to my favorite song.
    public static void rap() {
        System.out.println("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

Output:

```
Now this is the story all about how
My life got flipped turned upside-down

Now this is the story all about how
My life got flipped turned upside-down
```

Final Cookie Program

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies3 {
    public static void main(String[] args) {
        preheatOven();
        makeBatter();
        bake();        // 1st batch
        bake();        // 2nd batch
        decorate();
    }

    // Step 1: Preheat oven
    public static void preheatOven() {
        System.out.println("Preheat Oven to 375F.");
    }
    // Step 2: Make the cake batter.
    public static void makeBatter() {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
    }

    // Step 3: Bake a batch of cookies.
    public static void bake() {
        System.out.println("Set the timer for 8 min.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
    }

    // Step 4: Decorate the cookies.
    public static void decorate() {
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```

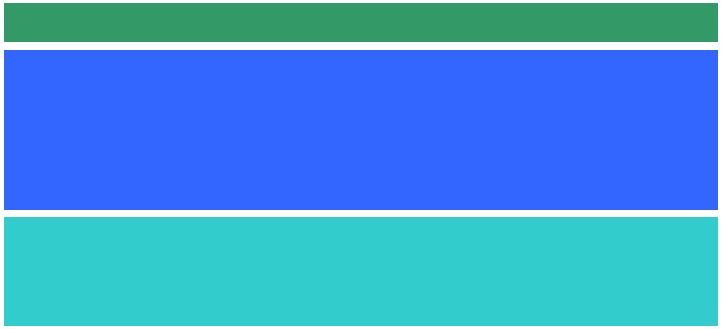
Examples: Modifying BakeCookies

- Bake three batches
- Change timer from 8 to 10 min

Summary: Why Methods?

- Capture structure of the program
  - `main` should be a good summary of the program

```
public static void main(String[] args) {
```



```
}
```

```
public static void main(String[] args) {
```

```
}

public static ...                (...) {
```

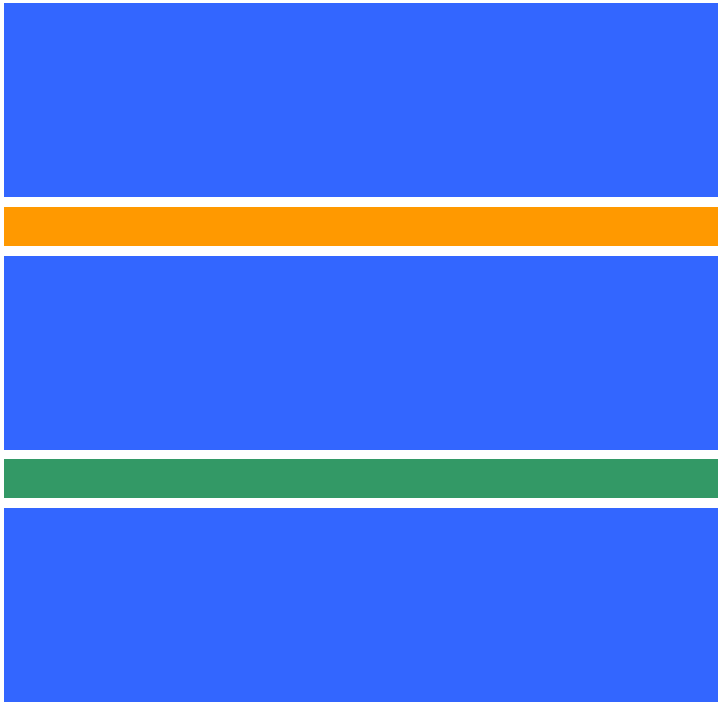
```
}

public static ...                (...) {
```

```
}
```

Summary: Why Methods?

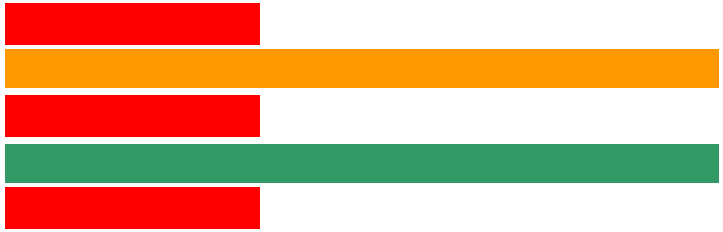
- Eliminate redundancy

```
public static void main(String[] args) {
```



```
}
```

```
public static void main(String[] args) {
```



```
}

public static ...                (...) {
```

```
}
```

Outline

- Admin.
- Java methods
  - Motivations
  - Syntax: declaring method
  - Method control flow

Method Calling Flow

- When a method A calls another method B, the program's execution...
  - "jumps" into method B, executing its statements, then
  - "jumps" back to method A at the point where the method was called.

Methods Calling Methods

```
public class MethodsExample {
    public static void main(String[] args) {
        message1();
        message2();
        System.out.println("Done with main.");
    }

    public static void message1() {
        System.out.println("This is message1.");
    }

    public static void message2() {
        System.out.println("This is message2.");
        message1();
        System.out.println("Done with message2.");
    }
}
```

Methods Calling Methods

```
public class MethodsExample {
    public static void main(String[] args) {
        message1();
        message2();
        System.out.println("Done with main.");
    }

    public static void message1() {
        System.out.println("This is message1.");
    }

    public static void message2() {
        System.out.println("This is message2.");
        message1();
        System.out.println("Done with message2.");
    }
}
```

- Output:

```
This is message1.
This is message2.
This is message1.
Done with message2.
Done with main.
```

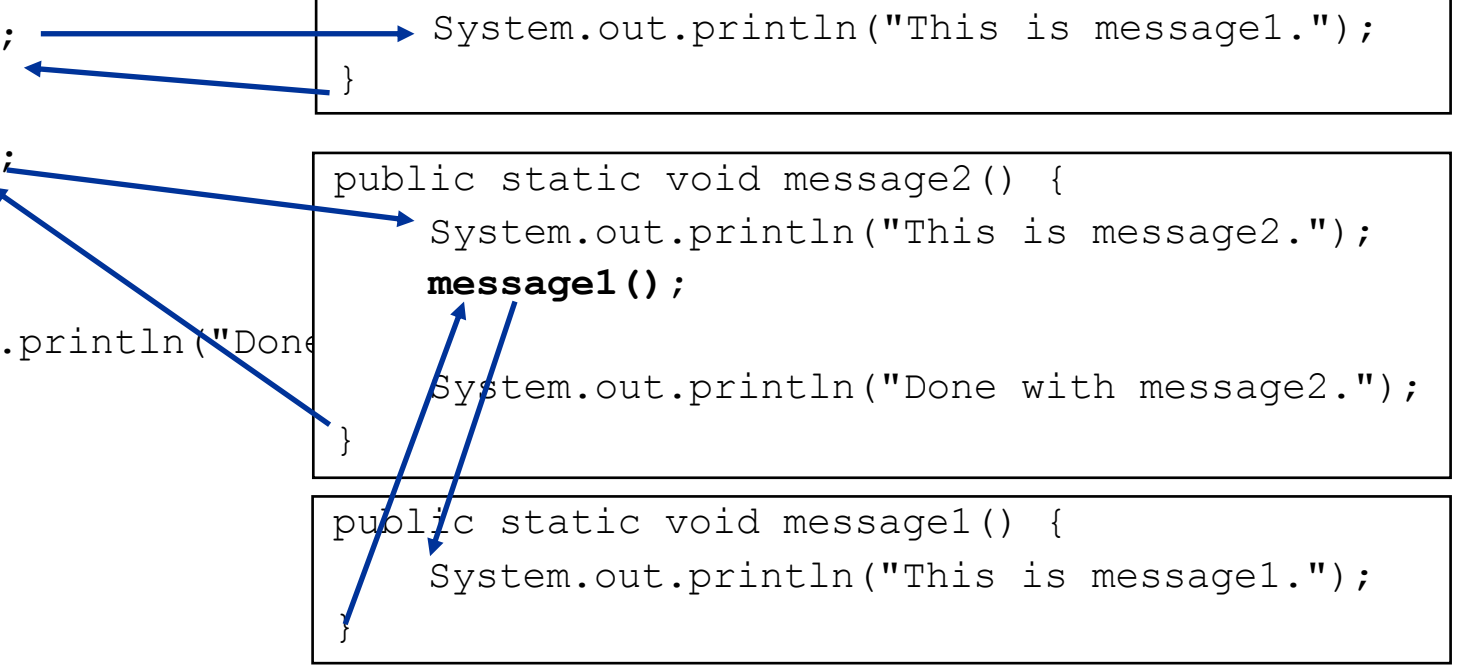
Methods Calling Methods

```
public class MethodsExample {
    public static void main(Stri
        message1();

        message2();


        System.out.println("Done
    }

    ...
}
```

```
public static void message1() {
    System.out.println("This is message1.");
}
```

```
public static void message2() {
    System.out.println("This is message2.");
    message1();

    System.out.println("Done with message2.");
}
```

```
public static void message1() {
    System.out.println("This is message1.");
}
```



Methods Calling Methods

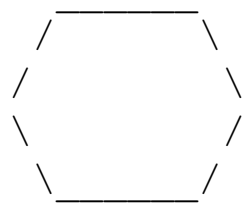
- Example: What is the output of Lullaby?

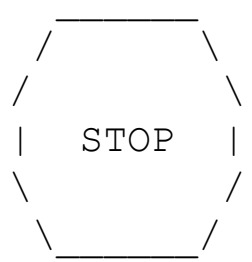
Outline

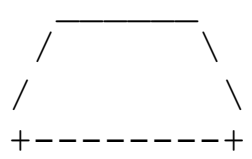
- Admin. and recap
- Java methods
  - Why methods?
  - Syntax: declaring method
  - Method control flow
  - Designing methods

Example

- Write a program to print these figures using methods.







Program version 1

```
public class Figures1 {
    public static void main(String[] args) {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("+--------+");
    }
}
```

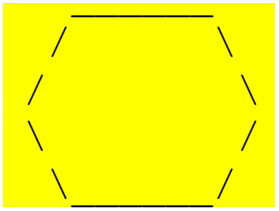
The code does not reflect structure.

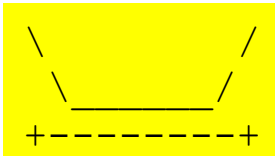
It has redundancy.

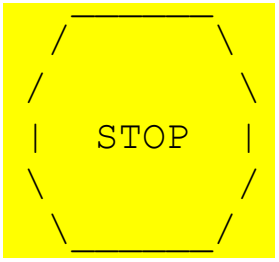
Method Design Techniques

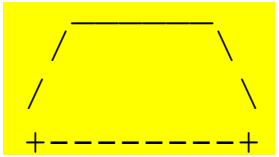
- A basic approach of designing methods, with consideration of structure and removing redundancy, is called top-down decomposition
  - dividing a problem into sub problems to be solved using methods

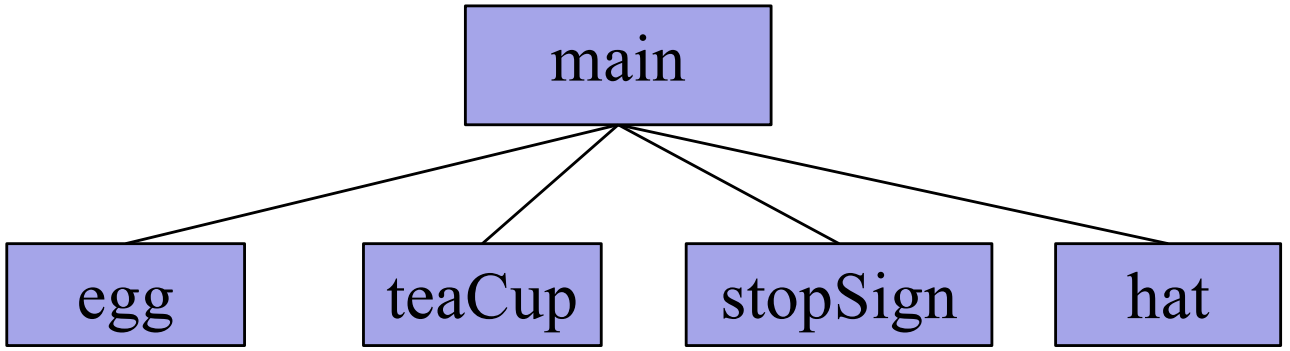
Top-Down Decomposition



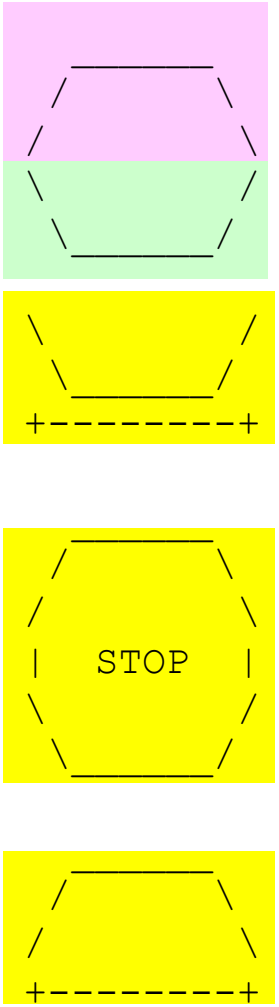


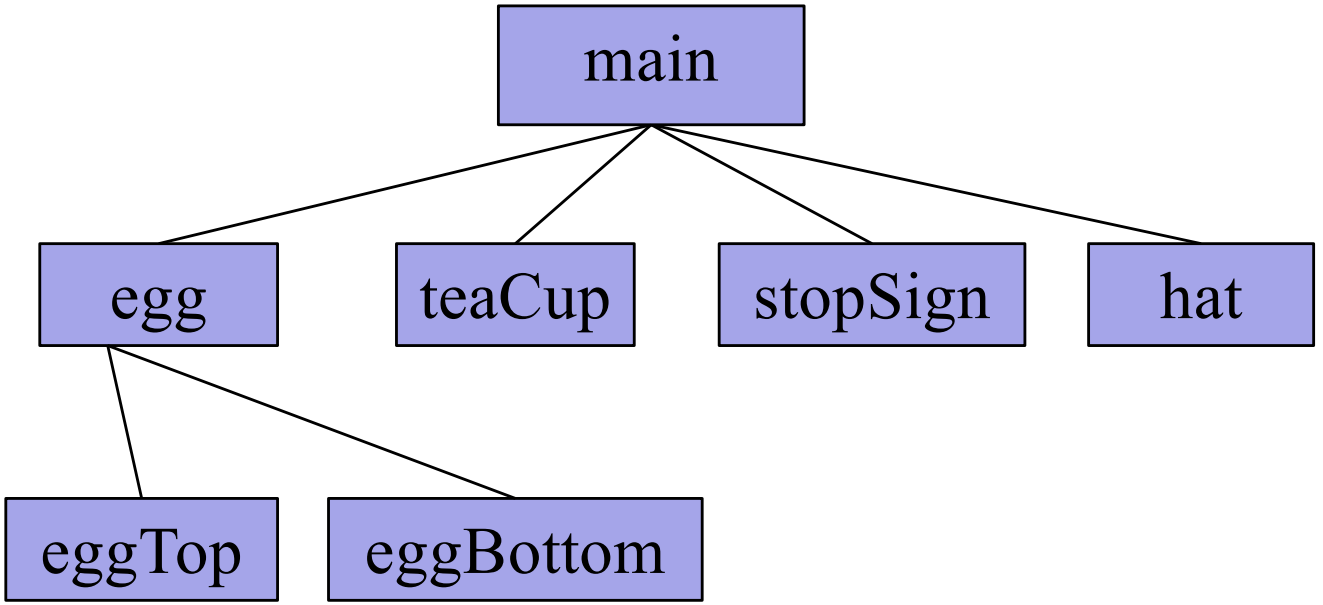




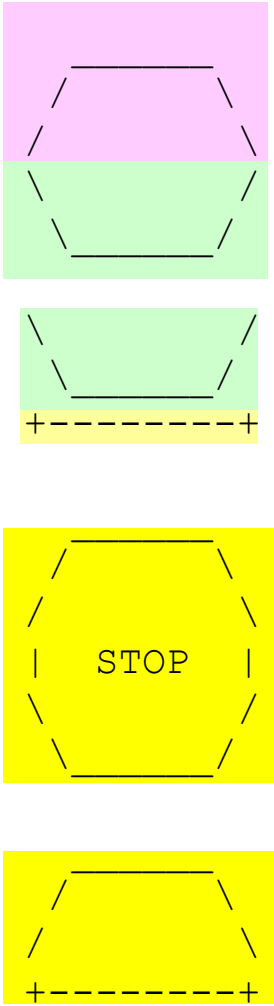


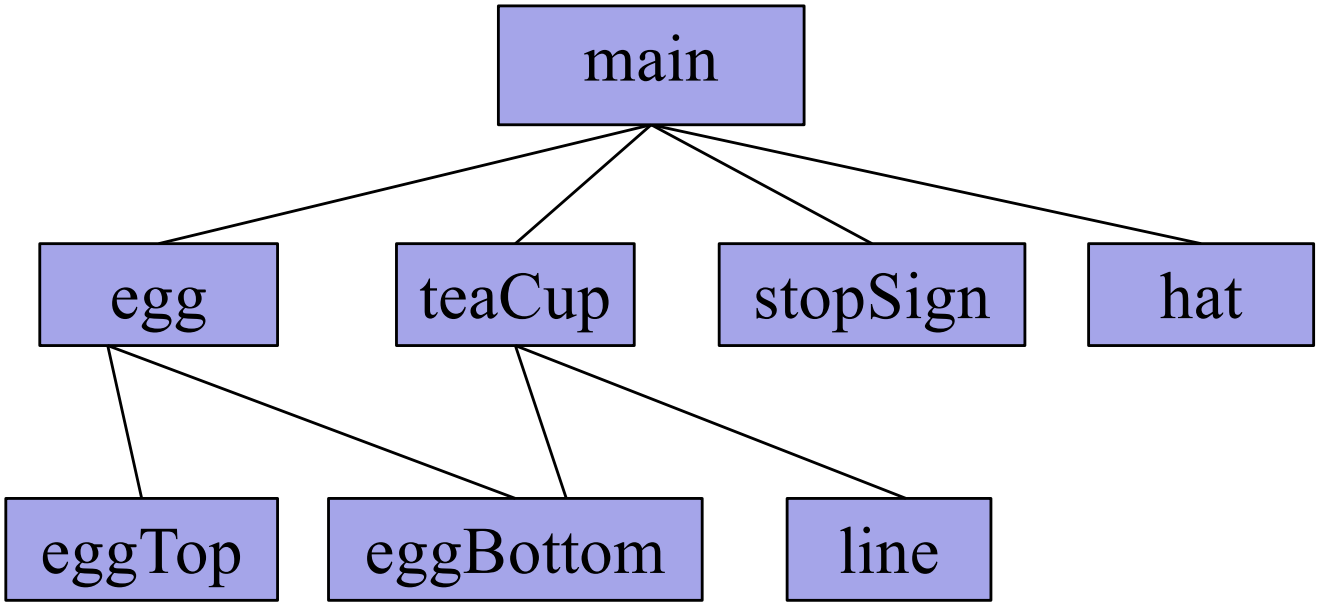
Top-Down Decomposition (egg)



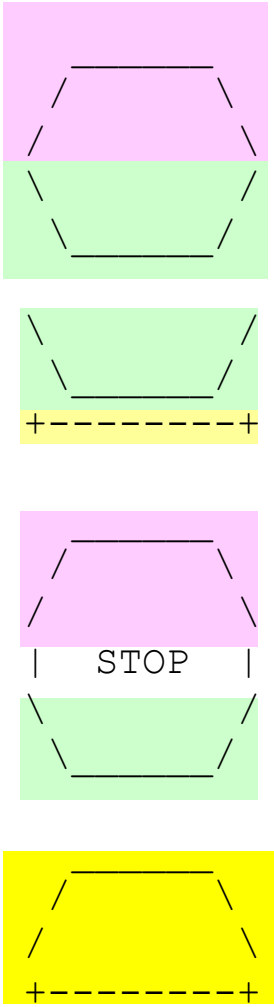


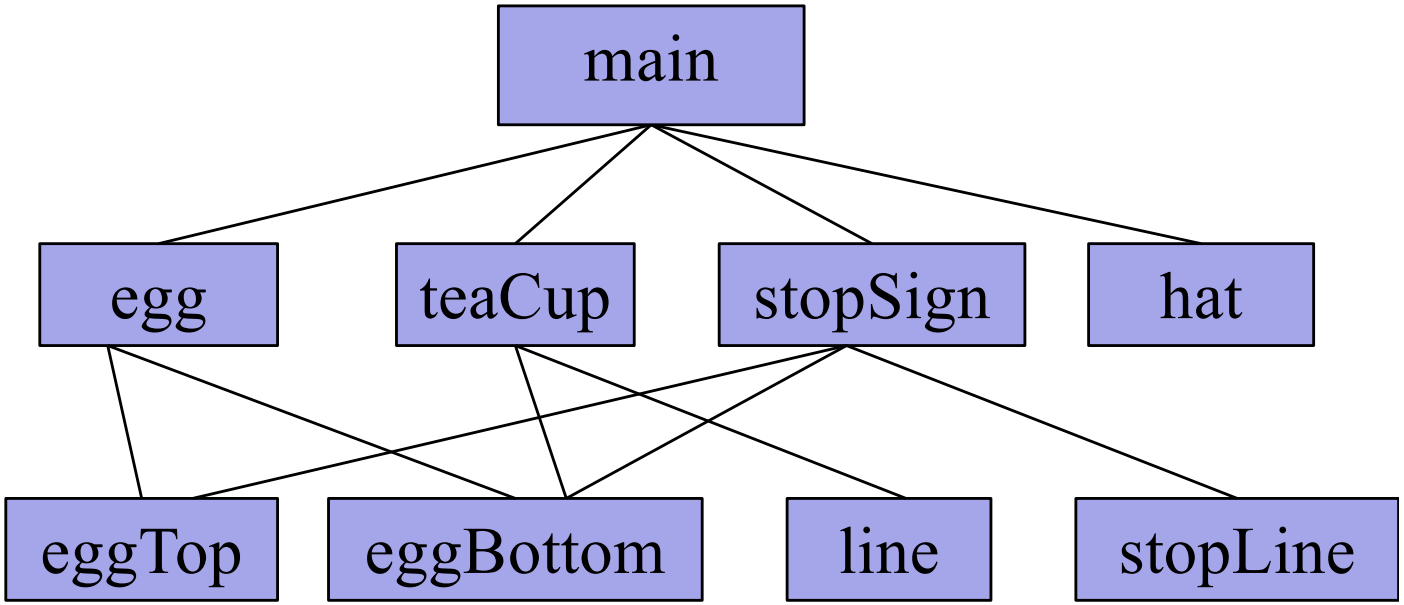
Top-Down Decomposition (teaCup)



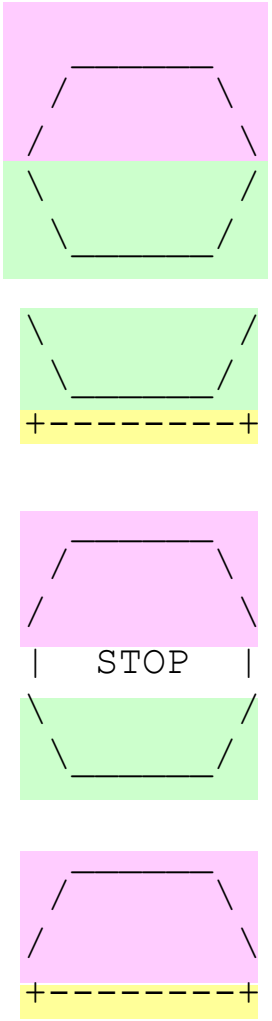


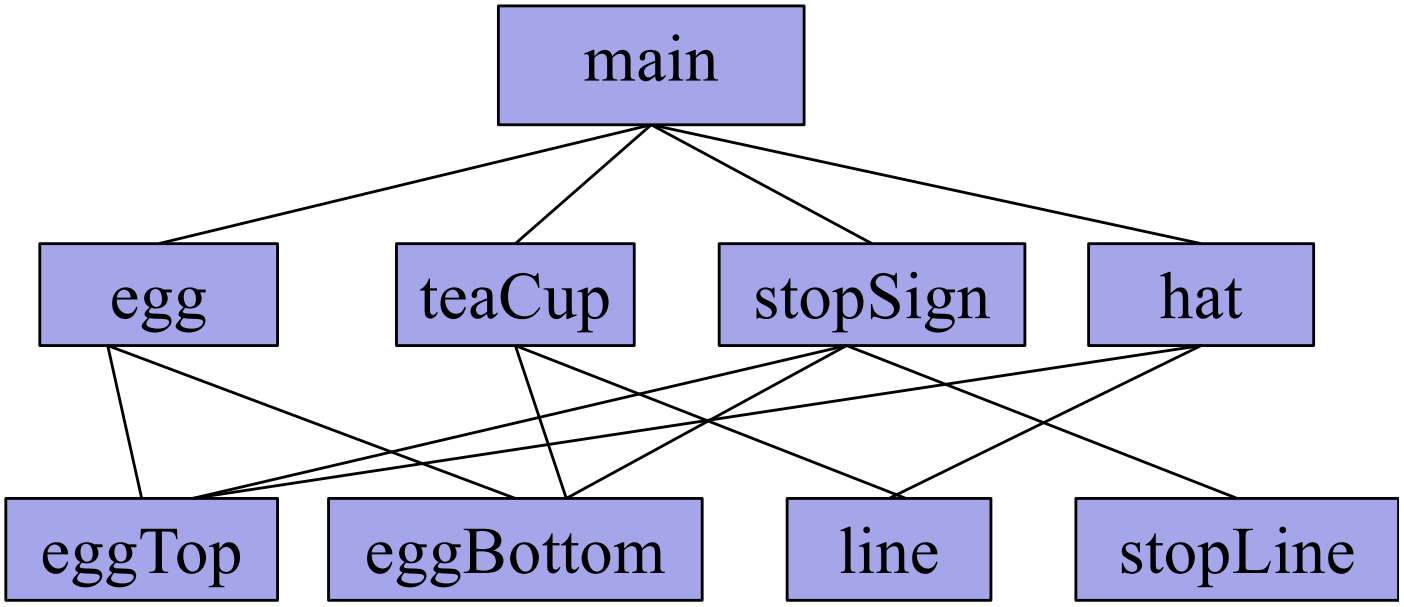
Top-Down Decomposition (stopSign)





Top-Down Decomposition (hat)





Q: What is a good order to implement/test the methods?

Structured Program version

```
// Prints several figures, with methods
// for structure and redundancy.
public class Figures3 {
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    // Draws the top half of an an egg figure.
    public static void eggTop() {
        System.out.println("  ______");
        System.out.println(" /______\\");
        System.out.println("/        \\");
    }

    // Draws the bottom half of an egg figure.
    public static void eggBottom() {
        System.out.println("\\        /");
        System.out.println(" \\______/");
    }

    // Draws a complete egg figure.
    public static void egg() {
        eggTop();
        eggBottom();
        System.out.println();
    }

    ...
```

Program version 3, cont'd.

```
    ...
    // Draws a line of dashes.
    public static void line() {
        System.out.println("+--------+");
    }

    // Draws a teacup figure.
    public static void teaCup() {
        eggBottom();
        line();
        System.out.println();
    }

    // Draws a stop sign figure.
    public static void stopSign() {
        eggTop();
        System.out.println("|  STOP  |");
        eggBottom();
        System.out.println();
    }

    // Draws a figure that looks sort of like a hat.
    public static void hat() {
        eggTop();
        line();
    }
}
```

A Word about Style

- Structure your code properly
- Eliminate redundant code

- Use comments to describe code behavior

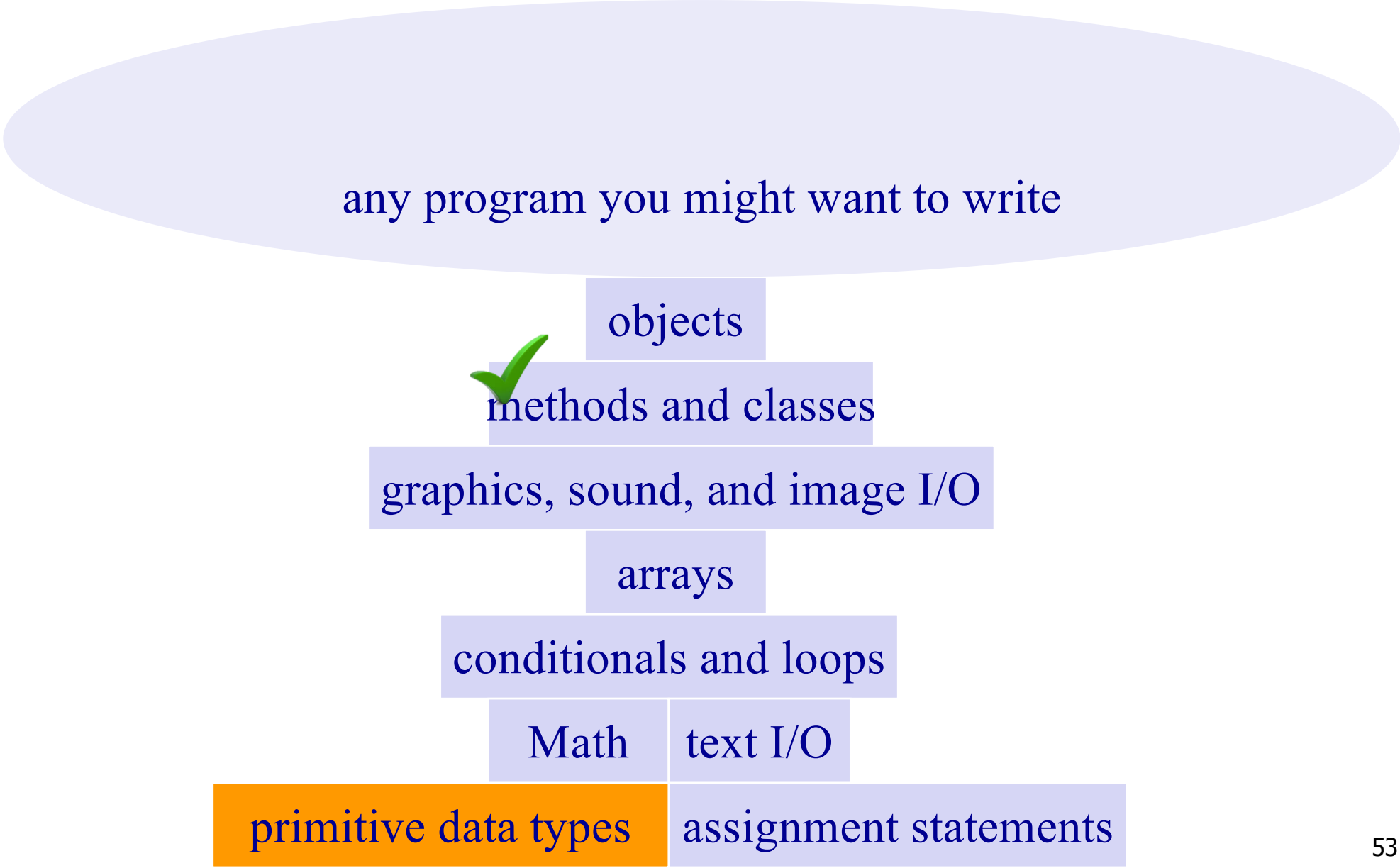
- Use spaces judiciously and **consistently**
- Indent properly

- Follow the naming conventions

Why Style?

- Programmers build on top of other's code all the time.
  - You shouldn't waste time deciphering what a method does.
- You should spend time on thinking or coding. You should **NOT** be wasting time looking for that missing closing brace.
- So code with style!

Foundational Programming Concepts



Outline

- Admin and recap
- Java methods
- Primitive data types
  - why data types

Memory

Primary storage area for programs and data

Main Memory

Also called RAM

RAM is divided into many cells; each cell can be identified by a numeric address

9278	00110000
9279	
9280	
9281	
9282	
9283	
9284	
9285	
9286	

Each memory cell has a set number of bits (usually 8 bits, or one *byte*); a bit can represent 2 values of 0 or 1)

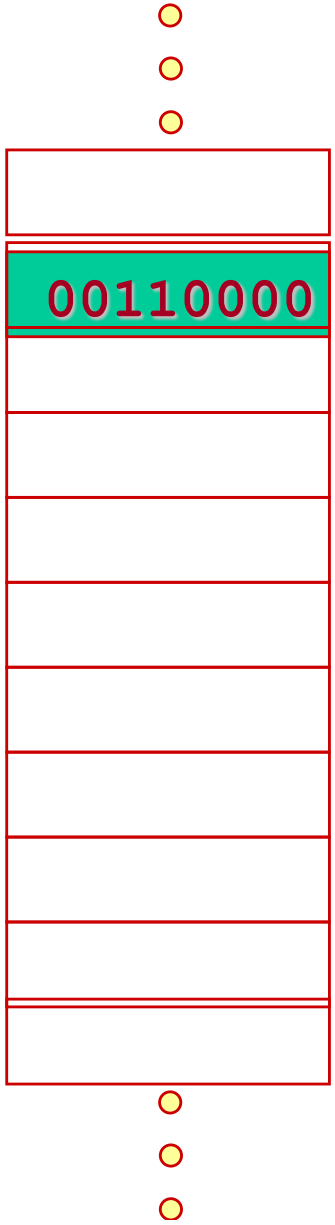
- how many possible values can a byte represent?

A computer can use multiple cells (e.g., 2 bytes) to store a value

- how many possible values can 2 bytes represent?

Variable

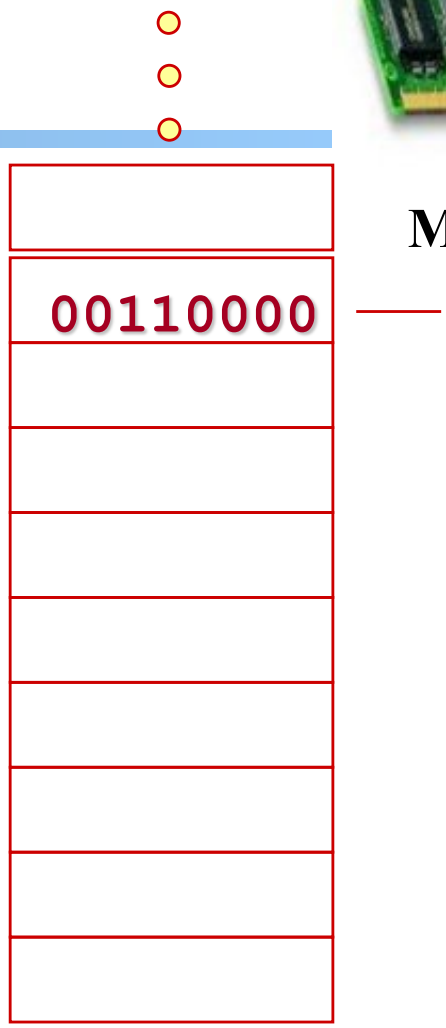
RAM is divided into many cells; each cell can be identified by a numeric address



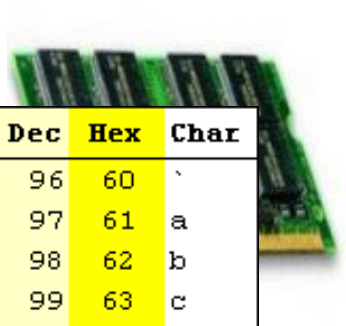
In high-level programming, instead of a numerical address, you can refer to a memory location by a name, say x. This is called a variable.

Problem

- What does the number (combination) stored at a given memory location represent?



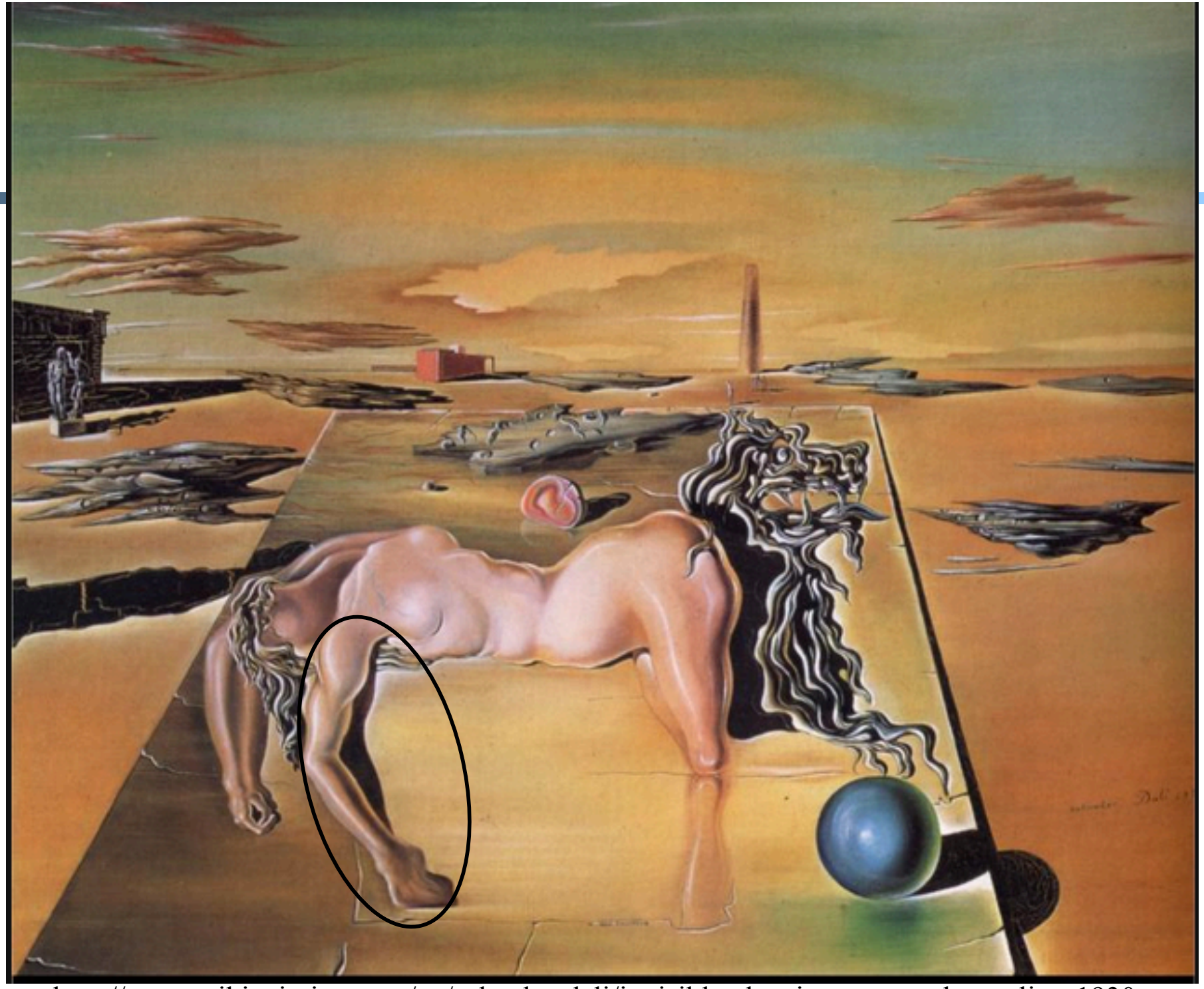
Two Examp



- 00110000 →
- 00110000 →

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

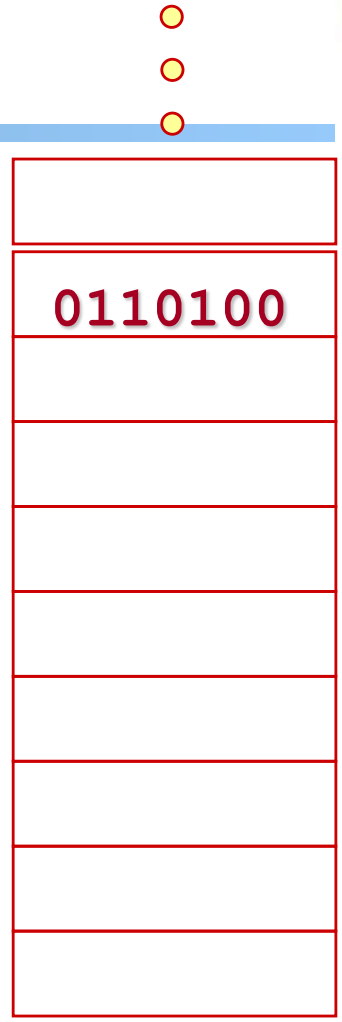
- Problem: How
  00110000 **st**



http://www.wikipaintings.org/en/salvador-dali/invisible-sleeping-woman-horse-lion-1930

Type System

- **type**: A category or set of values and operations defined on those values.
  - By specifying the type of a memory location, we know what the values represent
- Many languages ask the programmer to specify types
  - Examples: integer, real number, character



Main Memory

Variable and Type

- **Variable**: A piece of the computer's memory that is given a name and a type to store value of the type.

- Steps for using a variable:
  - *Declare* it - state its name and type
  - *Assign value* - initialize or update its value
  - *Use* it - print it or use it as part of an expression

Primitive Data Types

- There are eight (simple) primitive data types in Java
  - six numerical types (e.g., `int`, `double`)
    - for mathematical calculation
  - characters
    - for text processing
  - Boolean (logical) values
    - for decision making

Declaration

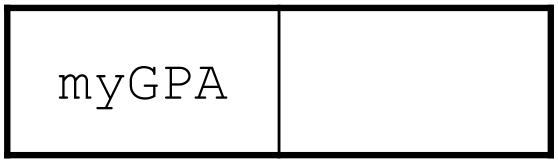
- **Variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- Syntax:

  <type> <name>;

  - `int x;`

  - `double myGPA;`



Assignment

- **Assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

  <name> = <expression>;

  - ```
    int x;
    x = 3;
    ```

x	3

  - ```
    double myGPA;
    myGPA = 1.0 + 2.25;
    ```

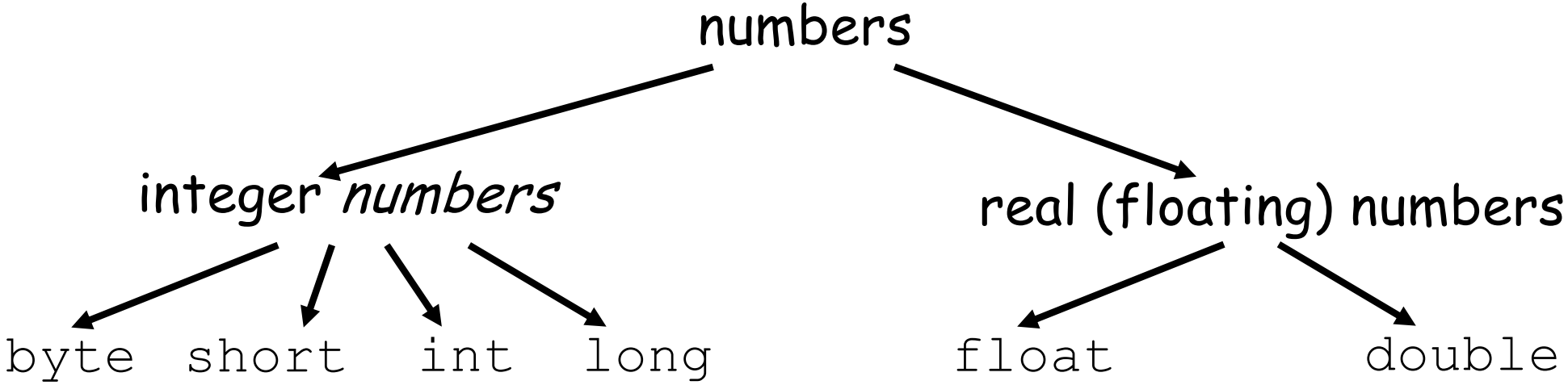
myGPA	3.25

- A variable can only store a value of its own type.

Outline

- Admin and recap
- Java methods
- Primitive data types
  - why data types?
  - storage and representation

Numeric Primitive Data Types



- The differences among the various numeric primitive types are their storage sizes and representation format, and hence the ranges & precision of the values they can store

Integer Numeric Data Types

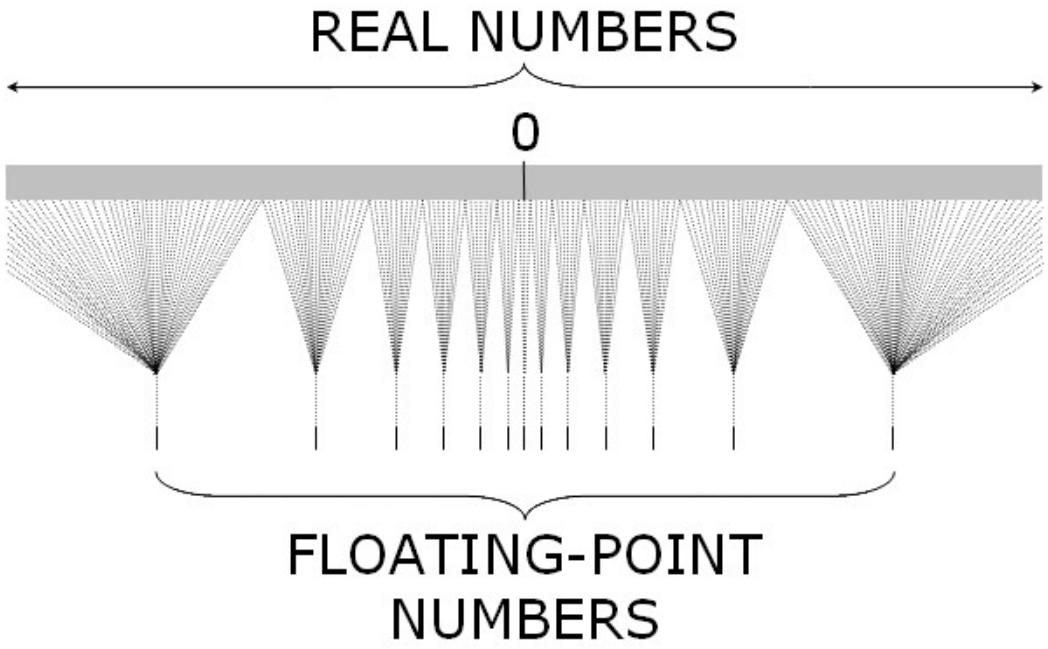
- Different integer numeric data types have different ranges and precision

Type	Storage	Min Value	Max Value
byte	1 byte	-128	127
short	2 bytes	-32,768	32,767
int	4 bytes	-2,147,483,648	2,147,483,647
long	8 bytes	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$

numbers with no fractional part

Real Numeric Data Types

Question: can computer store all real numbers in a range?



- Represented using the IEEE 754 format
  - with limited # of precision bits
  - See Precision.java

All Numeric Data Types

- Different integer numeric data types have different ranges and precision

Type	Storage	Min Value	Max Value	
byte	**1 byte**	**-128**	**127**	numbers with no fractional part
short	**2 bytes**	**-32,768**	**32,767**	
int	**4 bytes**	**-2,147,483,648**	**2,147,483,647**	
long	**8 bytes**	**< -9 x 10^{18}**	**> 9 x 10^{18}**	
float	**4 bytes**	**+/- 3.4 x 10^{38} with 7 significant digits**		IEEE 754 format
double	**8 bytes**	**+/- 1.7 x 10^{308} with 15 significant digits**		

Java Numerical Value and Type

- Java is a strongly typed language, i.e., every data item has a type
- An integer literal is by default of type `int`
  - that is, a literal number 4 in Java is of type `int`
  - to say that the number 4 is of type `long`, write 4l or 4L (4L is preferred over 4l since lower case "l" is hard to distinguish from 1)
- A real (floating point) literal (e.g., -1.23 6.12e23) is by default of type `double`
  - to say that the number 0.1 is of type `float`, write 0.1f or 0.1F

Questions

byte; short; int; long; float; double

Question: to represent the number of students at Xiamen University, which numeric data type variable do you use?

Question: to represent the world population, which numeric data type variable do you use?

Question: to represent your GPA, which numeric data type variable do you use?

Question: to represent a person's height in meters, which numeric data type variable do you use?

Question: to represent pi as 3.14159265359, which numeric data type variable do you use?

Real Life Example: Ariane 5

- Historical example: Ariane 5 explosion in 1996 (http://www.youtube.com/watch?v=kYUrqdUyEpI; http://www.ima.umn.edu/~arnold/disasters/ariane.html)

Real Life Example: Ariane 5

- Historical example: Ariane 5 explosion in 1996 (http://www.youtube.com/watch?v=kYUrqdUyEpI; http://www.ima.umn.edu/~arnold/disasters/ariane.html)

Real Life Example: Ariane 5

- Historical example: Ariane 5 explosion in 1996 (http://www.youtube.com/watch?v=kYUrqdUyEpI; http://www.ima.umn.edu/~arnold/disasters/ariane.html)

- Reason: range error
  - trying to store a 64-bit real number (a double) to a 16-bit integer led to the crash

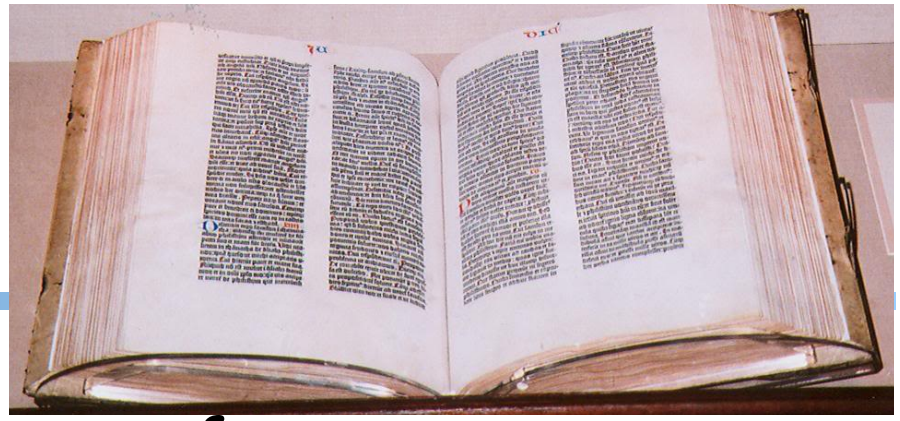
Real Life Example: Patriot Failure

- The Patriot Missile Failure in 1991
  - Perfect detection of a Scud missile, but the intercepting Patriot missed the target
- Reason: precision error
  - a computer cannot represent 0.1 precisely; for a 24-bit floating point number they used, it is off by 0.000000095.
  - After 100 hours in operation, it is off by 0.34 seconds (=0.000000095*100 hours * 60 min/hour * 60 sec/min * 10), leading to an error of about 600 meters (http://www.ima.umn.edu/~arnold/disasters/patriot.html)

In the Movie

http://www.youtube.com/watch?v=G_wiXgRWrIU

Characters



- A `char` is a single character from *a character set*
- A *character set* is an ordered list of characters; each character is given a unique number
- Character literals are represented in a program by delimiting with single quotes:

```
'a'   'X'   '7'   '$'   ','  '\n'
```

Java Character Set

- Java uses the Unicode character set, a superset of ASCII
  - uses sixteen bits (2 bytes) per character, allowing for 65,536 unique characters
  - it is an international character set, containing symbols and characters from many languages
  - code chart can be found at:

http://www.unicode.org/charts/

Boolean

- A `boolean` value represents logical value: true or false
- The keywords `true` and `false` are the only valid values for a boolean type
- A boolean can also be used to represent any two states, such as a light bulb being on or off

Outline

- Admin and recap
- Java Methods
- Primitive data types
  - why data types?
  - storage and representation
  - operations

Data Type and Operations

- A type defines not only the storage/representation but also the allowed and meaning (semantics) of operations
  - Discussions: reasonable operations that can be performed on two operands
    - Integers: i1 ? i2
    - Strings: s1 ? s2
    - Characters: c1 ? c2

Data Type and Operations

type	set of values	literal values	operations
`char`	characters	`'A'` `'@'`	compare (more details later on +-)
`String`	sequences of characters	`"Hello"` `”112 is fun"`	concatenate +
`int`	integers	`17` `12345`	compare add +, sub -, multiply *, divide /, modulus %
`double`	floating-point numbers	`3.1415` `6.022e23`	compare add +, sub -, multiply *, divide /, modulus %
`boolean`	truth values	`true` `false`	==, !=, and &&, or ||, not !

Data Type and Operations

- Most operations (+, -, *, /) are intuitive and similar to our daily-life use
- Perhaps a first major surprise in learning programming is that the result of an operation depends on the data type

3 + 5	v.s.	"3" + "5"
3 / 5	v.s.	3.0 / 5.0

See TypeDep.java

Interpretation

You should think that there are multiple versions of the same operator, each for a type, e.g.,

- +int +string …
- /int /double …

Integer Division with /

- When we divide integers, the result is an integer (the fractional part is discarded)
  - `14 / 4` **is** `3`, **not** `3.5`

```
   3                  4
4 ) 14            10 ) 45
   12                40
    2                 5
```

- More examples:
  - `32 / 5` **is** **6**
  - `8 / 10` **is** **0**
  - `156 / 100` **is** **1**
  - Dividing by 0 causes an error when your program runs.

Integer Remainder with %

- The % operator computes the remainder from integer division.
  - `14 % 4` **is** `2`
  - `218 % 5` **is** `3`

```
    3
4 ) 14
    12
     2
```

```
    43
5 ) 218
    20
    18
    15
     3
```

Practice (offline):

45 % 6

2 % 2

8 % 20

11 % 0

Obtain last digit of a number: `230857 % 10` **is** `7`

Obtain last 4 digits: `230857 % 10000` **is** `857`

See whether a number is odd: `7 % 2` **is** `1`, `42 % 2` **is** `0`

Outline

- Admin and recap
- Primitive data types
  - why data types?
  - storage and representation
  - operations
  - expressions

Evaluating Arithmetic Expression

- Arithmetic operators can be combined into complex arithmetic expressions
  - `(7 + 2) * 6 / 3`

- The evaluation order of the operators in an arithmetic expression is determined by a well-defined precedence order
  - Remember?
    - ***Pretty Please My Dear Aunt Sally***

Operator Precedence Rules

- Generally operators evaluate left-to-right.
  `1 - 2 - 3` **is** `(1 - 2) - 3` **which is** `-4`

- But `* / %` have a higher level of precedence than `+ -`
  `1 - 3 * 4` **is** `-11`

- Parentheses can force a certain order of evaluation:
  `(1 + 3) * 4` **is** `16`

- Spacing does not affect order of evaluation
  `1+3 * 4-2` **is** `11`