Introduction to Computational Thinking

Lecture #6:

Methods with Return;

Text Input/Output (Scanner); Object (briefly);

Boolean Expressions; Nested if/else;

Qiao Xiang, Qingyu Song

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

11/5/2025

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.

Outline

- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules
    - Overloaded methods
    - Formal arguments are local variables
    - Primitive types use value semantics

Outline

- Admin and recap
- Method details
  - Method w/ return

Different Styles of Methods

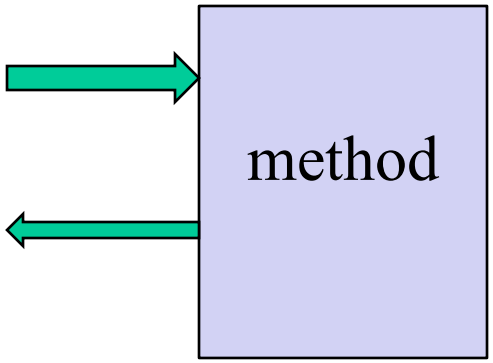
"Action-oriented methods":
External effects (print, drawing, audio), e.g.,
drawCar(0,0,10), drawCar(10, 10, 20), …

"Answer-oriented methods":
e.g., how much is sqrt(2), sqrt(10)?

"Mixed methods": do both

Method with Return

- Math methods are good examples of "answer-oriented" methods: they do useful work by *returning values*



Example: `Math` Methods

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Math Methods

- Simply calling math methods produces no visible result.
  - `Math.pow(3, 4);  // no output`
- To see the result, we must print or store the returned value:
  - `System.out.println( Math.pow(3, 4) ); // 81.0`
  - `double result = Math.pow(3, 4);`
  - `System.out.println(result);  // 81.0`

Why return and not print?

- It might seem more useful for the `Math` methods to print their results rather than returning them. Why don't they?

- Answer: Returning is more flexible than printing.
  - We can compute several things before printing:

```
double pow1 = Math.pow(3, 4);
double pow2 = Math.pow(10, 6);
System.out.println("Powers are " + pow1 + " and " +
pow2);
```

  - We can combine the results of many computations:

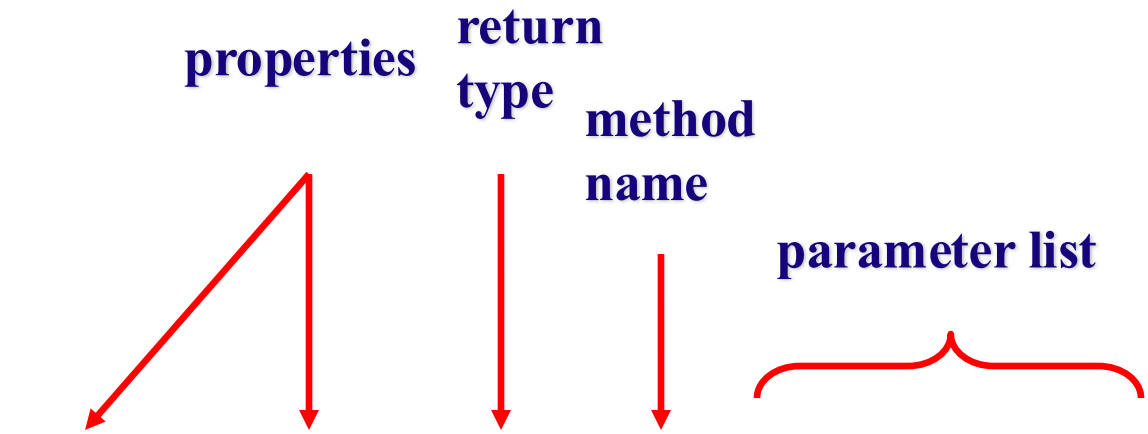
```
double k = 13 * Math.pow(3, 4) + 5 - Math.sqrt(17.8);
```

http://download.oracle.com/javase/6/docs/api/java/lang/Math.html

Math Questions

- Evaluate the following expressions:
  - `Math.abs(-1.23)`
  - `Math.toRadians( 180 )`
  - `Math.round(Math.PI) + Math.round(Math.E)`
  - `Math.abs(Math.min(-3, -5))`
  - `Math.random() //(between 0-1)`
- Consider an `int` variable named `age`.
  - What expression would replace negative ages with 0?
    - `Math.max(age, 0)`
  - What expression would cap the maximum age to 25?
    - `Math.min(age, 25)`

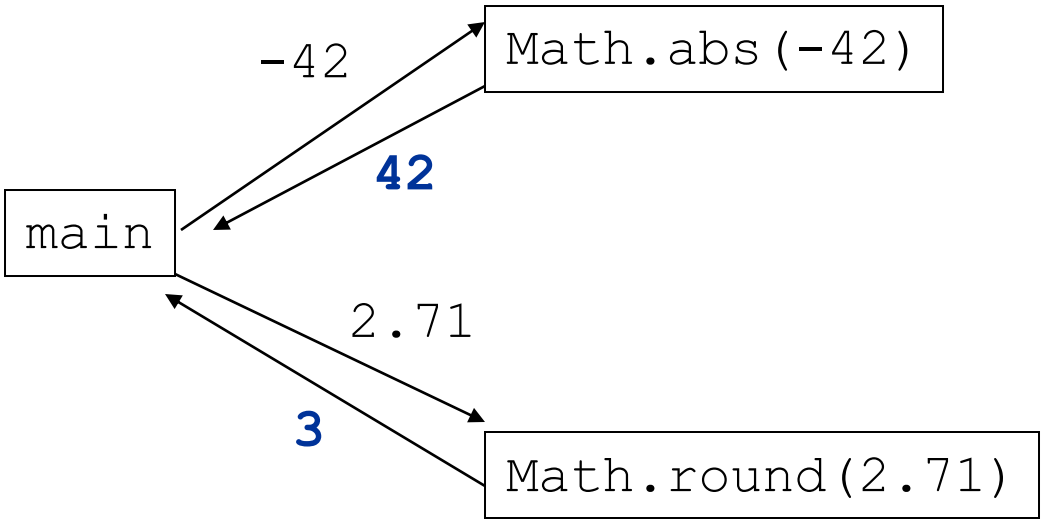
Defining a Method Returning a Value




```
public static type name( parameters ) {
    statements;
    ...
    return expression;
}
```

Return vs Paremeter

- Return is the opposite of a parameter:
  - Parameters send information *in* from the caller to the method.
  - Return value sends information *out* from a method to its caller.



Exercise: Fahrenheit -> Celsius

- Write method f2c that converts from Fahrenheit to Celsius, both in double

Return Example

```
// Converts degrees Fahrenheit to Celsius.
public static double f2c(double degreesF) {
    double degreesC = (degreesF - 32) * 5.0 / 9.0 ;
    return degreesC;
}
```

- You can shorten the example by returning an expression:

```
public static double fToC(double degreesF) {
    return (degreesF - 32) * 5.0 / 9.0 ;
}
```

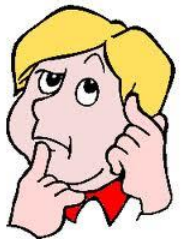
A Common Error

- Many people incorrectly think that a `return` statement sends a variable's name back to the calling method.

```
public static void main(String[] args) {
    fToc(60);
    System.out.println("60F = " + result);
}

public static double fToC(double degreesF) {
    double result = 5.0 / 9.0 * (degreesF - 32);
    return result;
}
```

// ERROR: result not defined



Fixing the Common Error

- Instead, returning sends the variable's *value* back.
  - The returned value must be stored into a variable or used in an expression to be useful to the caller.

```
public static void main(String[] args) {
  double c = fToC(65);
  System.out.println("65F = " + c + "C");
  System.out.println("Again, 65F = " + fToC(65) + "C");
}

 public static double fToC(double degreesF) {
    double result = 5.0 / 9.0 * (degreesF - 32);
    return result;
 }
```

Outline

- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules

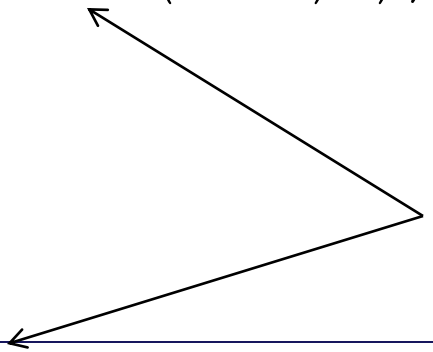
Summary: Method Definition

- We have learnt all of Methods.
- Why define methods?
  - Denote structure, eliminate redundancy
  - A method with parameters solves an entire class of similar problems
  - A method with return gives back an answer on a question

Method Invocation "Puzzle"

Many methods in Math has multiple definitions:
https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

```
System.out.print( Math.round(10.3) );
```




```
// Math.round() has two definitions

// definition 1
static long round(double a)

// definition 2
static int round(float a)
```

Method Definition and Invocation Rules

- Definition rule:
  - You can define multiple methods with the same name in a class. This is called method overloading
  - To distinguish different overloaded methods, these methods must have different signatures
    - The signature is the sequential list of the type of each parameter

- Invocation rule:
  - Java compiler picks the method with the best match in signature, allowed by implicit conversion.

Overloaded Methods

Version 1: signature: int

```
double tryMe (int x)
{
    return x + .375;
}
```

Version 2: signature: int_double

```
double tryMe (int x, double y)
{
    return x * y;
}
```

Version 3: signature: double_int

```
double tryMe (double x, int y)
{
    return x * y;
}
```

Invocation

```
result = tryMe (25, 4.32)
```

Version 4: signature: double_double

```
double tryMe (double x, double y)
{
    return x * y;
}
```

Overloading Picks the Best Match allowed by Implicit Conversion

```
double tryMe ( int x )
{
    return x + 5;
}
```

```
double tryMe ( double x )
{
    return x * .375;
}
```

```
double tryMe (double x, int y)
{
    return x + y;
}
```

Which tryMe will be called?

```
tryMe( 1 );

tryMe( 1.0 );

tryMe( 1.0, 2);

tryMe( 1, 2);

tryMe( 1.0, 2.0);
```

Overload Matching only Signature

```
int x = (int)Math.round(10.3);
```

```
int x = Math.round(10.3);
```

ERROR: Type mismatch.

Best match. (Double if default)

I know 10 will fit as an int: how do I change from long to int?

```
// Math.round() has two definitions

// definition 1
static long round(double a)

// definition 2
static int round(float a)
```

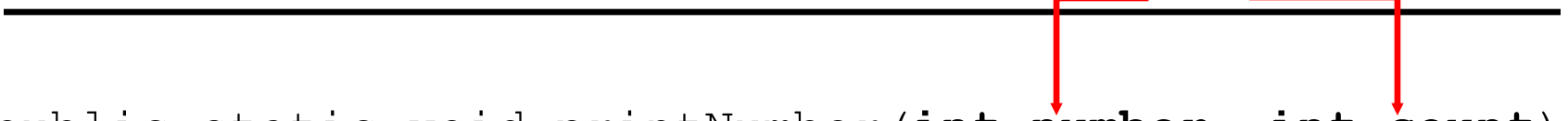
Outline

- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules
    - Overloaded methods
    - Formal arguments are local variables

Method Invocation and Parameter Passing

- Corresponding actual *argument* in the invocation is assigned to the corresponding *formal argument*

```
int line = 3;
printNumber(line-1,5);
```



```
public static void printNumber(int number, int count)
{
    // equiv: number = 2; count = 5;
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

Method Invocation and Parameter Passing

- In Java, a formal argument is a local variable of a method
- The formal argument and the actual argument are different variables, with different memory locations, even if they have the same name.

Exercise: "Parameter Mystery"

```
public class ParameterMystery {
    public static void main(String[] args) {
        int x = 9;
        int y = 2;
        int z = 5;

        mystery(z, y, x);

        mystery(y, x, z+y);
    }


    public static void mystery(int x, int y, int z) {
        System.out.println(z + " and " + (y - x));
    }
}
```

Outline

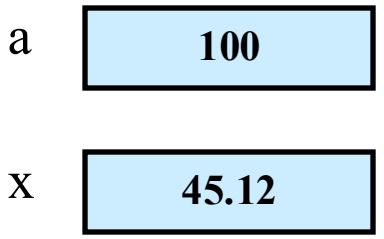
- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules
    - Overloaded methods
    - Formal arguments are local variables
    - Primitive types use value semantics

Method Invocation and Parameter Passing

- When a primitive variable is passed as the actual argument to a formal argument, the value is copied
  - Value copying implies value semantic
  - Implication: modifying the parameter inside the method will not affect the variable passed in.

Value Semantics

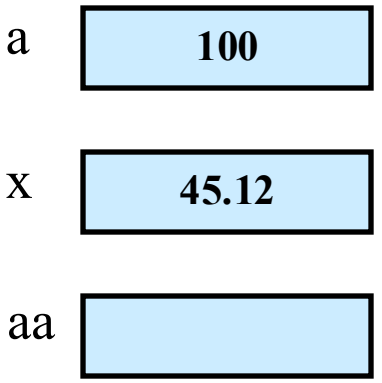
```
int a = 100;
double x = 45.12;
```



A value variable stores a value of the type of the variable.

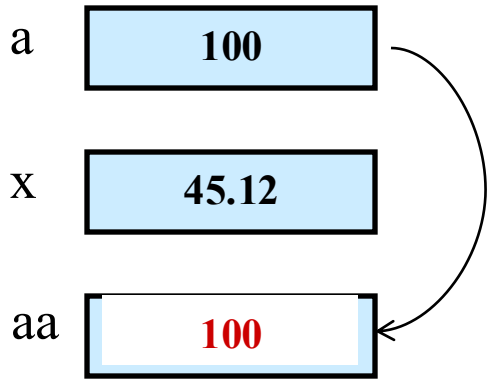
Value Variables

```
int a = 100;
double x = 45.12;
int aa;
```



Value-Variable Assignment

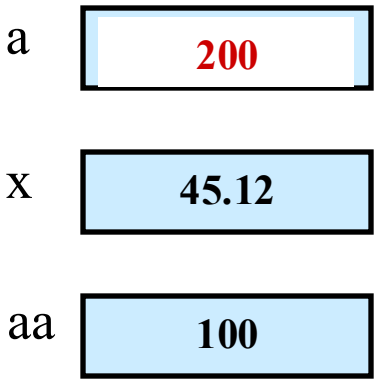
```
int a = 100;
double x = 45.12;
int aa;
aa = a;
```



An assignment of one value variable to another value variable copies the value.

Value-Variable Assignment

```
int a = 100;
double x = 45.12;
int aa;
aa = a;
a = 200;
```



Change the value of one value variable will not change the other.

Exercise: "Parameter Mystery"

```
public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}

public static void main(String[] args) {
    int x = 23;
    strange(x);
    System.out.println("2. x = " + x);
    ...
}
```

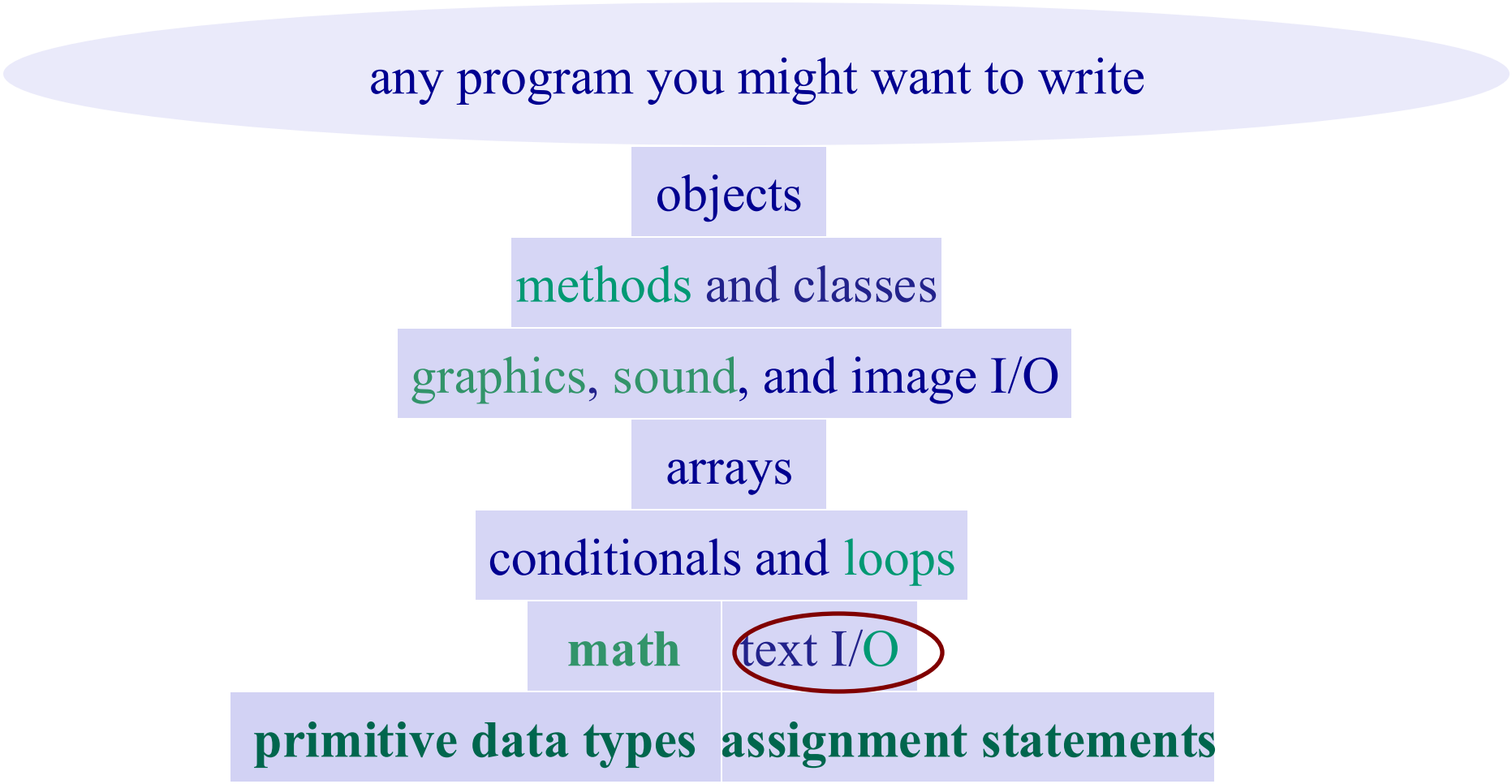
Output:

```
1. x = 24
2. x = 23
```

Outline

- Admin and recap
- Method w/ return
- Summary of method definition and invocation rules
  - overloaded methods
  - formal arguments are local variables
  - primitive types use value semantics
- Text I/O
  - Input: basic Scanner input
  - Output: basic printf and String.format

Foundational Programming Concepts



Motivation: CarLaunch with User Input

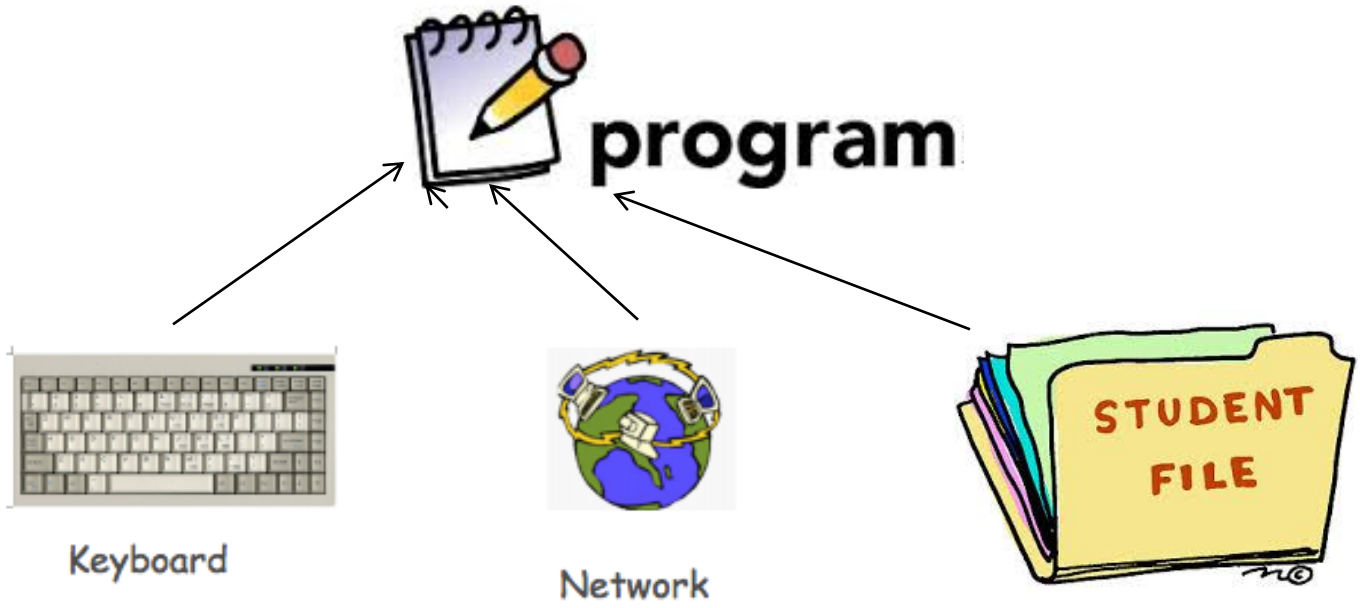
- Extend the CarLaunch program to get input from user on initial parameters:
  - h1, v1x, v1y (Car)
  - h2, v2x, v2y (Angry Bird)
  - sound file to play
- Benefit:
  - Change program parameters without need to change the program and recompilation
- Issue: Interactive programs can be tricky: users are unpredictable and may misbehave.
- Java user input is based on the `Scanner` class

Some `Scanner` Methods

Method	Description
`nextInt()`	Returns an `int` from source
`nextDouble()`	Returns a `double` from source
`next()`	Returns a one-word `String` from source
`nextLine()`	Returns a one-*line* `String` from source

Problem of Defining `Scanner`

- It is common that the same program reads input simultaneously from multiple sources:
  - `System.in` (the opposite of `System.out`)
  - Files, strings, web sites, databases, ...



Design Option I

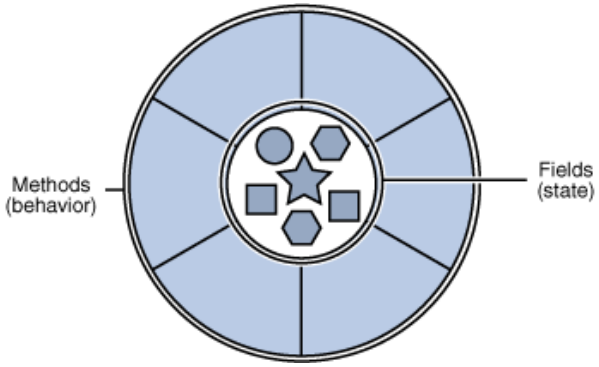
Method
`Scanner.nextInt(<src>)`
`Scanner.nextDouble(<src>)`
`Scanner.next(<src>)`
`Scanner.nextLine(<src>)`

Problem: A lot of redundancy in program,

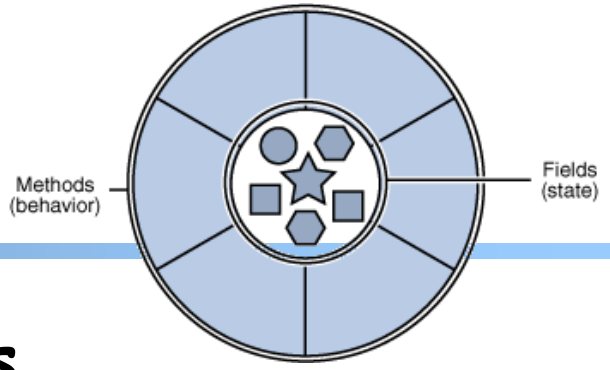
```
h1  = Scanner.nextDouble("System.in");
v1x = Scanner.nextDouble("System.in");
v1y = Scanner.nextDouble("System.in");
h2  = Scanner.nextDouble("System.in");
v2x = Scanner.nextDouble("System.in");
v2y = Scanner.nextDouble("System.in");
soundFile = Scanner.next("System.in");
```

Design Option II: Objects (briefly)

- **object:** An entity that contains both data (state) and behavior.
  - *Data (state)*
    - variables inside the object to remember state
  - *behavior*
    - methods offered by the object
  - You interact with the methods; most data (state) are hidden in the object (think of object methods as methods with memory for now).



Constructing Objects



- An object is created from a **class**
- Constructing (creating) an object by calling the constructor method:

```
Type objectName = new Type(parameters);
```

- Calling an object's method:

```
objectName.methodName(parameters);
```

Using Scanner

```
import java.util.Scanner;
…

Scanner console = new Scanner(System.in);

// Typically print a prompt
System.out.print("How old are you? ");

int age = console.nextInt();

System.out.println("You typed " + age);
```

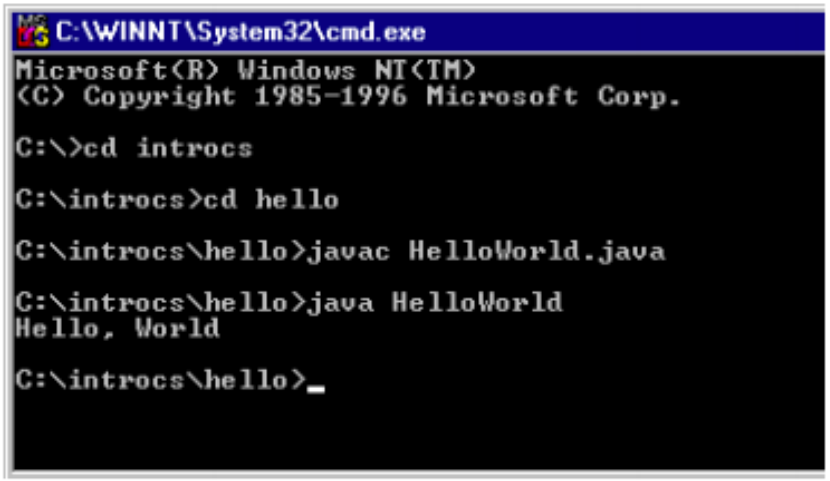
No need to specify that it is from standard input (keyboard)

Create an object, which remembers that it is for standard input

Scanner using System.in as Input

- Using System.in is to interact using the Terminal:

Mac OS X



Microsoft Windows

Scanner Example

```
import java.util.Scanner;    // so that I can use Scanner

public class ScannerInputExample {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Which year will you graduate? ");
        int gYear = console.nextInt();

        int rYear = gYear - 2025;
        System.out.println(rYear + " more years at XMU! ");
    }
}
```

gYear	2026
rYear	1

- Console (user input underlined):

```
Which year will you graduate?
1 more years at XMU!
```

2026

ScannerInputExample.java

Scanner Example 2

- The Scanner can read multiple values from one line.

```
import java.util.Scanner;   // so that I can use Scanner

public class ScannerMultiply {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Please type two numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();

        int product = num1 * num2;
        System.out.println("The product is " + product);
    }
}
```

- Output (user input underlined):

```
Please type two numbers: 8 6
The product is 48
```

ScannerMultiply.java