COMP 110-001
Mid-Term Review

Yi Hong

May 27, 2015

Announcement

- Midterm on Friday, May 29
  - Closed books, no notes, no computer

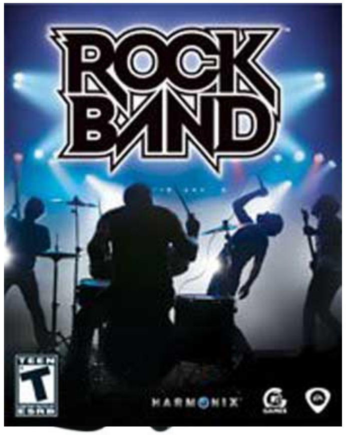
Today

- A whirlwind tour of almost everything we have covered so far

- You should start preparing for mid-term if you haven’t
- Finish the mid-term practice before Thursday
- Review slides and textbook
- Review your lab / assignment code

Hardware vs. Software

- Hardware - physical machine
  - CPU, Memory
- Software - programs that give instructions to the computer
  - Windows XP, Games, Eclipse



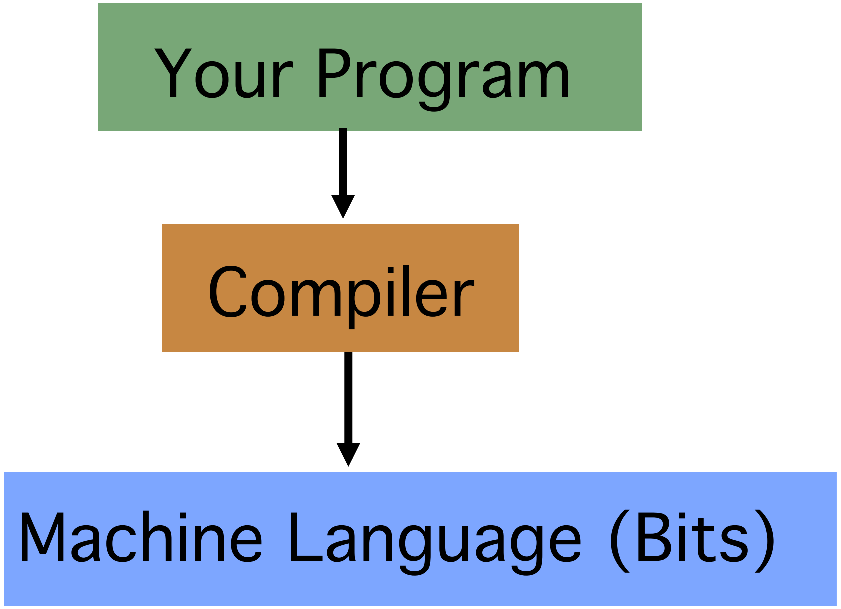
Hardware

- CPU – the “brain” of your computer
- Memory – stores data for the computer
  - How much the “brain” can remember
  - Main memory: RAM
  - Auxiliary memory: Hard Drive

Memory

- Measured in bytes
- 1 byte = 8 bits
- Bit is either 0 or 1
- Language of the computer is in bits

Programming Languages



High-level language
(human readable)

Low-level language
(computer readable)

Algorithms and Pseudocode

- Algorithm – a set of instructions for solving a problem

- Pseudocode – combination of code and English used to express an algorithm **before** writing algorithm into code
  - We can also use flow-chat to write pseudocode

Variables

- Used to store data in a program
- The data currently in a variable is its **value**
- Name of variable is an **identifier**
  - **Letters, digits, underscore**
  - Cannot start with digits
- Can change value throughout program
- Choose variable names that are meaningful!

How to Use Variables

- **Declare** a variable
  - `int number;`
- **Assign** a value to the variable
  - `number = 37;`
- **Change** the value of the variable
  - `number = 513;`

Keywords

- Reserved words with predefined meanings
- You *cannot* name your variables keywords
- if, else, return, new

Data Type

- What kind of value the variable can hold
- Two kinds of types.
  - Primitive type - indecomposable values
    - Names begin with lowercase letters
    - int, double, char, float, byte, boolean, and others
  - Class type - objects with both data and methods
    - Names by convention begin with uppercase letter
    - Scanner, String, Student

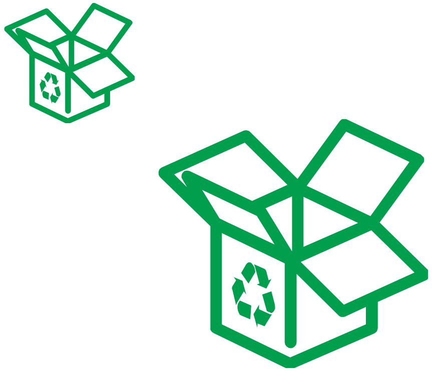
Assignment Statements

- Change a variable’s value
- Syntax
  - variable = expression;
- Example
  - sleepNeeded = 8;
  - sleepDesired = sleepNeeded * 2;

Assignment Compatibilities

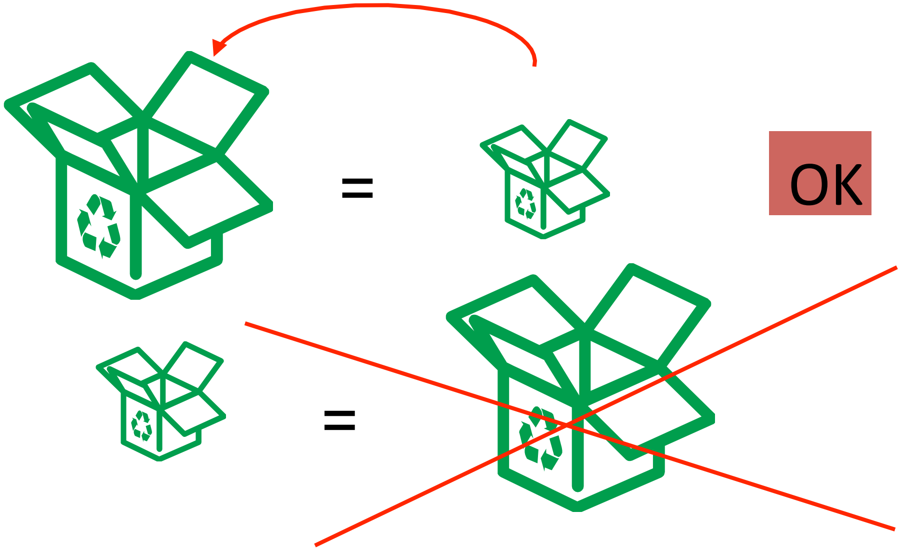
int x = 5;

double y = 12.7;



byte → short → int → Long → float → double

y = x; →



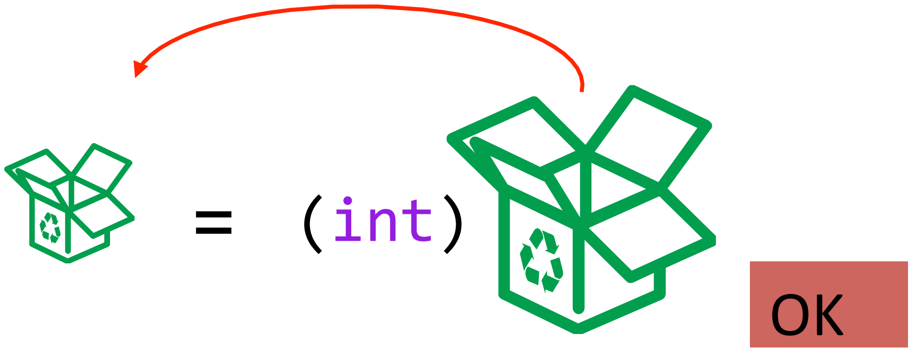
OK

x = y; →

Not OK

Type Casting

x = (int) y; →



= (int)

OK

Arithmetic Operators

- Unary operators
  - +, -, ++, --, !
- Binary arithmetic operators
  - *, /, %, +, -
    - rate*rate + delta
    - 1/(time + 3*mass)
    - (a - 7)/(t + 9*v)

Modular Arithmetic: %

- Remainder
- 7 % 3 = 1 (7 / 3 = 2, remainder 1)
- 8 % 3 = 2 (8 / 3 = 2, remainder 2)
- 9 % 3 = 0 (9 / 3 = 3, remainder 0)

Parentheses and Precedence

- Expressions inside parentheses evaluated first
  - (cost + tax) * discount
  - cost + (tax * discount)
- Precedence rules

Highest Precedence

  - First: the unary operators +, -, !, ++, and --
  - Second: the binary arithmetic operators *, /, %
  - Third: the binary arithmetic operators + and –

Lowest Precedence

Errors

- *Syntax error* – grammatical mistake in your program
    - Java will not compile programs with syntax error
- *Run-time error* – an error that is detected during program execution
    - E.g., exceptions during execution
- *Logic error* – a mistake in a program caused by the underlying algorithm

Strings

- A string (lowercase) is a sequence of characters
  - “Hello world!”
  - “Enter a whole number from 1 to 99.”
- String (capital S) is a class in Java, not a primitive type

String

```
String animal = “aardvark”;
System.out.println(animal);
```

```
aardvark
```

String Concatenation

```
String animal = “aardvark”;
String sentence;
sentence = “My favorite animal is the ” + animal;
```

```
My favorite animal is the aardvark
```

String’s Methods

- myString.length();
- myString.equals(“a string”);
- myString.toLowerCase();
- MyString.indexOf(‘ ’);
- myString.trim();
- …

- For other methods, check Java API

String Indices

U	N	C		i	s		G	r	e	a	t
0	1	2	3	4	5	6	7	8	9	10	11

```
String output = myString.substring(1, 8);
```

String Indices

U	N	C		i	s		G	r	e	a	t
0	1	2	3	4	5	6	7	8	9	10	11

```
String output = myString.substring(1, 8);
```

Escape Characters

\”	Double quote
\’	Single quote
\\	Backslash
\n	New line
\r	Carriage return
\t	Tab

Keyboard Input

```
Scanner keyboard = new Scanner(System.in);
int num = keyboard.nextInt();
```

Comments

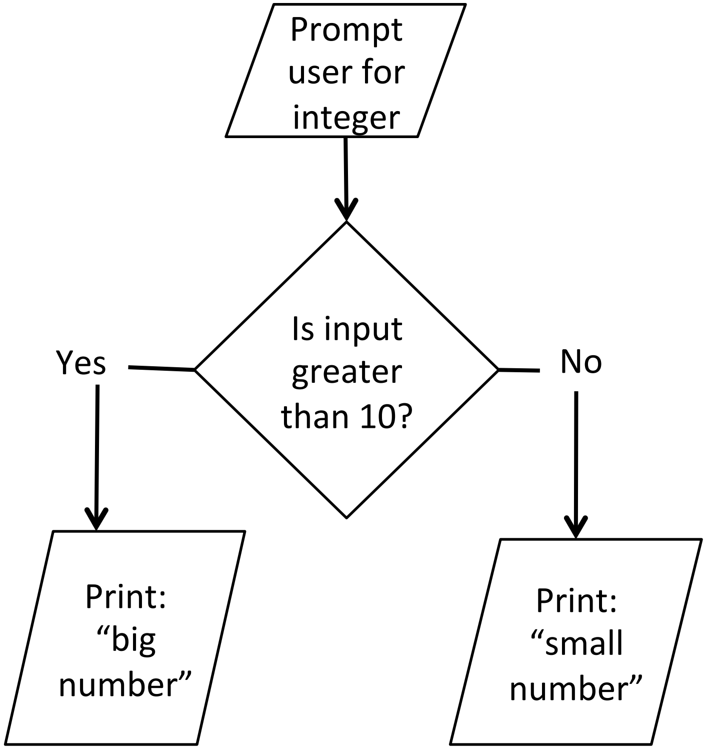
```
// this is a comment

/* This is also
   a comment.
  it ends
  here --->*/
```

Boolean Expressions

- An expression that is either true or false
- Examples:
  - It is sunny today (true)
  - 10 is larger than 5 (true)
  - Today is Saturday (false)

if/else Statements



```
import java.util.*;

public class FlowChart
{
    public static void main(String[] args)
    {
        System.out.println("Give me an integer:");
        Scanner keyboard = new Scanner(System.in);
        int inputInt = keyboard.nextInt();

        if (inputInt > 10)
        {
            System.out.println("big number");
        }
        else
        {
            System.out.println("small number");
        }
    }
}
```

If-else-if for Multi-Branch Selections

```
if ( case1 ) {
    // branch 1

} else if ( case2) {
    // branch 2

} else if ( case3 ) {
   …
   …
} else {
    …
}
```

```
if (year==1) {
    System.out.println(“Freshman”);
} else if (year==2) {
    System.out.println(“Sophomore”);
} else if (year==3) {
    System.out.println(“Junior”);
} else {
    System.out.println(“Senior”);
}
```

Java Comparison Operators for Primitive Values

`==`	Equal to
`!=`	Not equal to
`>`	Greater than
`>=`	Greater than or equal to
`<`	Less than
`<=`	Less than or equal to

Example expressions:

```
variable <= 6
myInt > 5
5 == 3
```

The result is a boolean value (true/false)

Boolean Type

- Can be either true or false

```
boolean sunny = true;
boolean cloudy = false;

if (sunny || cloudy)
{
  // walk to school
}
```

&&, || operators

- AND

```
if ((temperature > 50) && (temperature < 75))
{
  // walk to school
}
```

- OR

```
if (sunny || cloudy)
{
  // walk to school
}
```

The ! (NOT) operator

- !true is false
- !false is true
- Example: walk to school if it is NOT cloudy

```
if (!cloudy)
{
  // walk to school
}
```

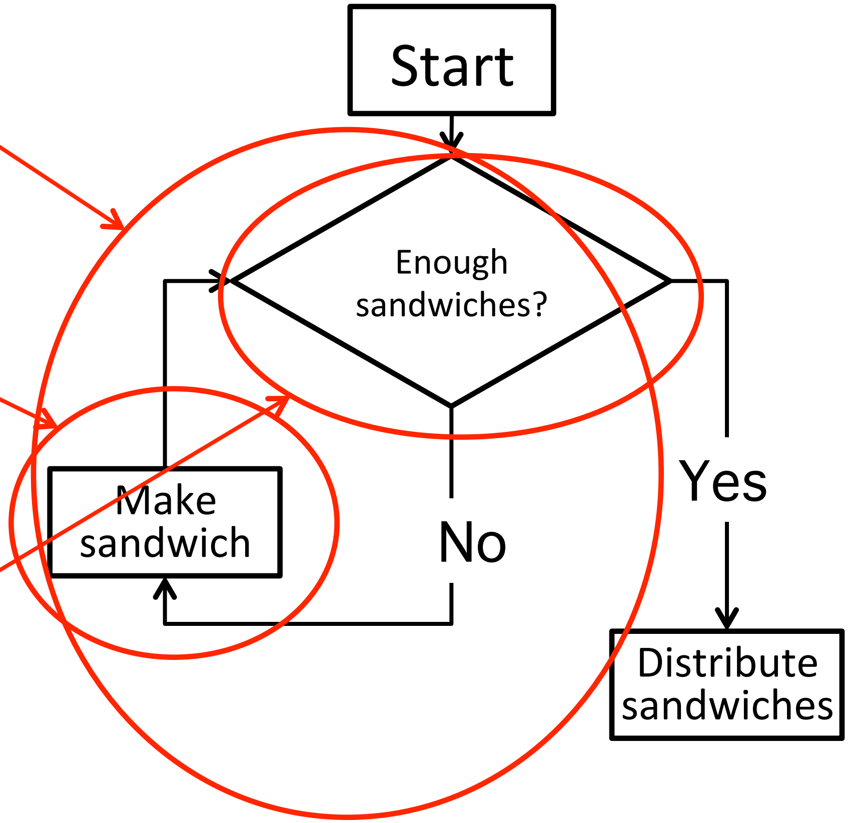
Loops

- Loop: part of a program that repeats

- Body: statements being repeated

- Iteration: each repetition of body

- Stopping condition

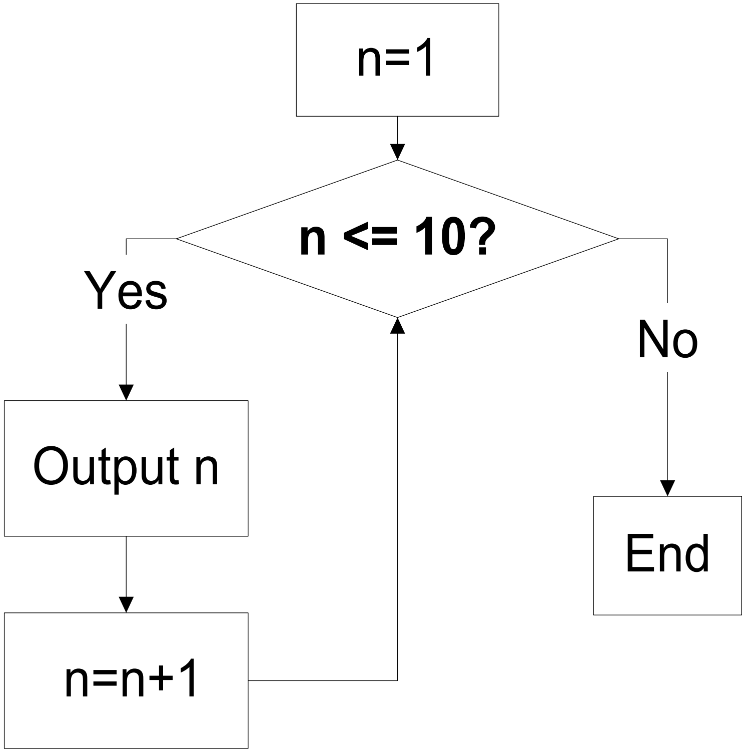


Types of Loops

- while
  - Safest choice
  - Not always the best
- do-while
  - Loop iterates AT LEAST once
- for
  - Similar to while, but often more convenient syntax
  - Most useful when you have a known number of iterations you need to do

Using a while Loop

```
int n = 1;
while (n <= 10)
{
    System.out.println(n);
    n = n + 1;
}
```



Using a for Loop

```
int n;

for (n = 1; n <= 10; n++)
{
    System.out.println(n);
}
```

Infinite Loop Example

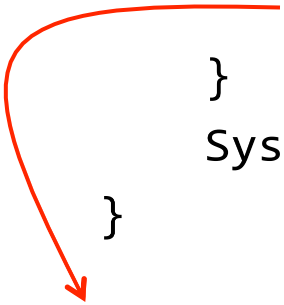
```
int n;

for (n = 1; n <= 10; n = 0)
{
    System.out.println(n);
}
```

The break statement

```
for (int item = 1; item <= 5; item++)
{
    System.out.print(“Enter cost of item #” + item + “: $”);
    amount = keyboard.nextDouble();
    total = total + amount;
    if (total >= 100)
    {
        System.out.println(“You spent all your money.”);
        break;
    }
    System.out.println(“Your total so far is $” + total);
}

System.out.println(“You spent $” + total);
```



Ending a Loop

- Count-controlled loops
  - If you know the number of loop iterations
  - for (count = 0; count < iterations; count++)

- User-controlled loops
  - Change the value of control variable
    - E.g., *Ask-before-iterating, or sentinel value (if user input is smaller than 0)*
    - E.g., booleans, matching is found

Nested Loops Example

```
for (int i = 1; i<10; i++) {
    for (int j = 1; j<=i; j++) {
        System.out.print( i + ”*” + j + “=“ + (i * j) + “\t“);
    }
    System.out.println();
}
```

Inner loop

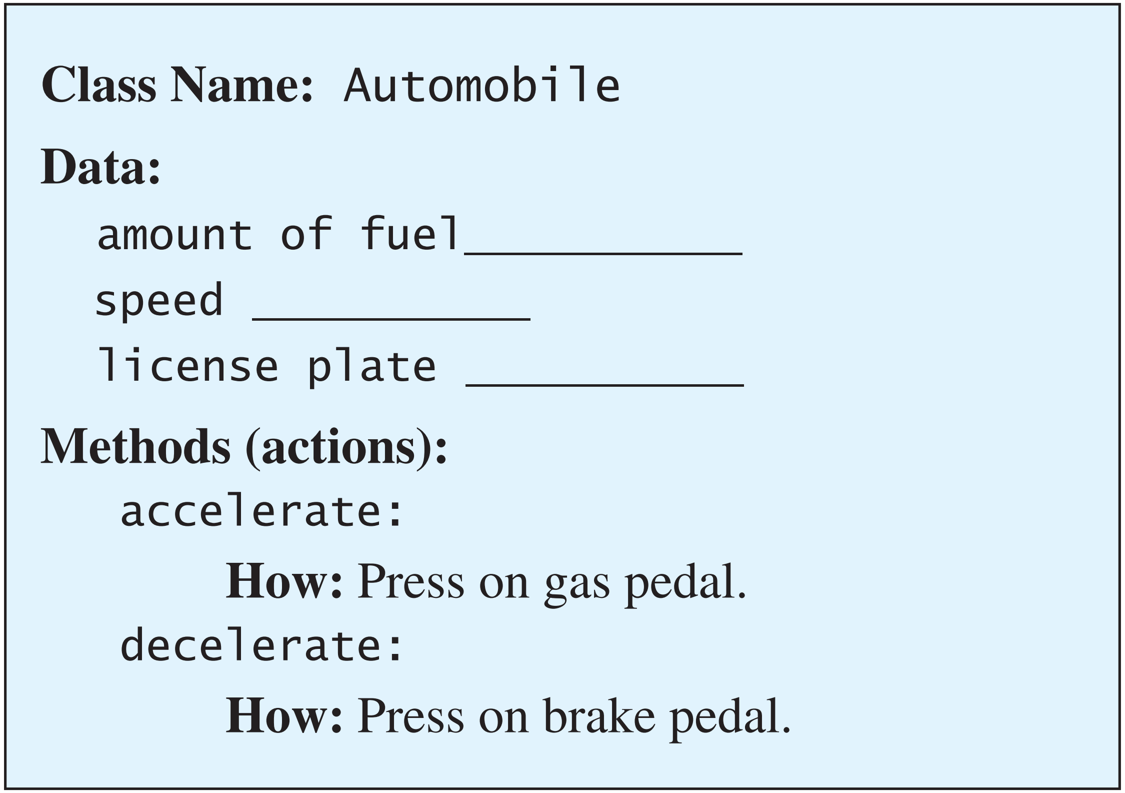
Outer loop

Classes, Objects, and Methods

- Class: a definition of a kind of object
- Object: an instance of a class
  - Contains instance variables (data) and methods
- Methods
  - Methods that return a value
  - Methods that return nothing

Class

- A *class* is the definition of a kind of object
  - A blueprint for constructing specific objects



Class Name: `Automobile`

Data:

`amount of fuel`____________

`speed` ____________

`license plate` ____________

Methods (actions):

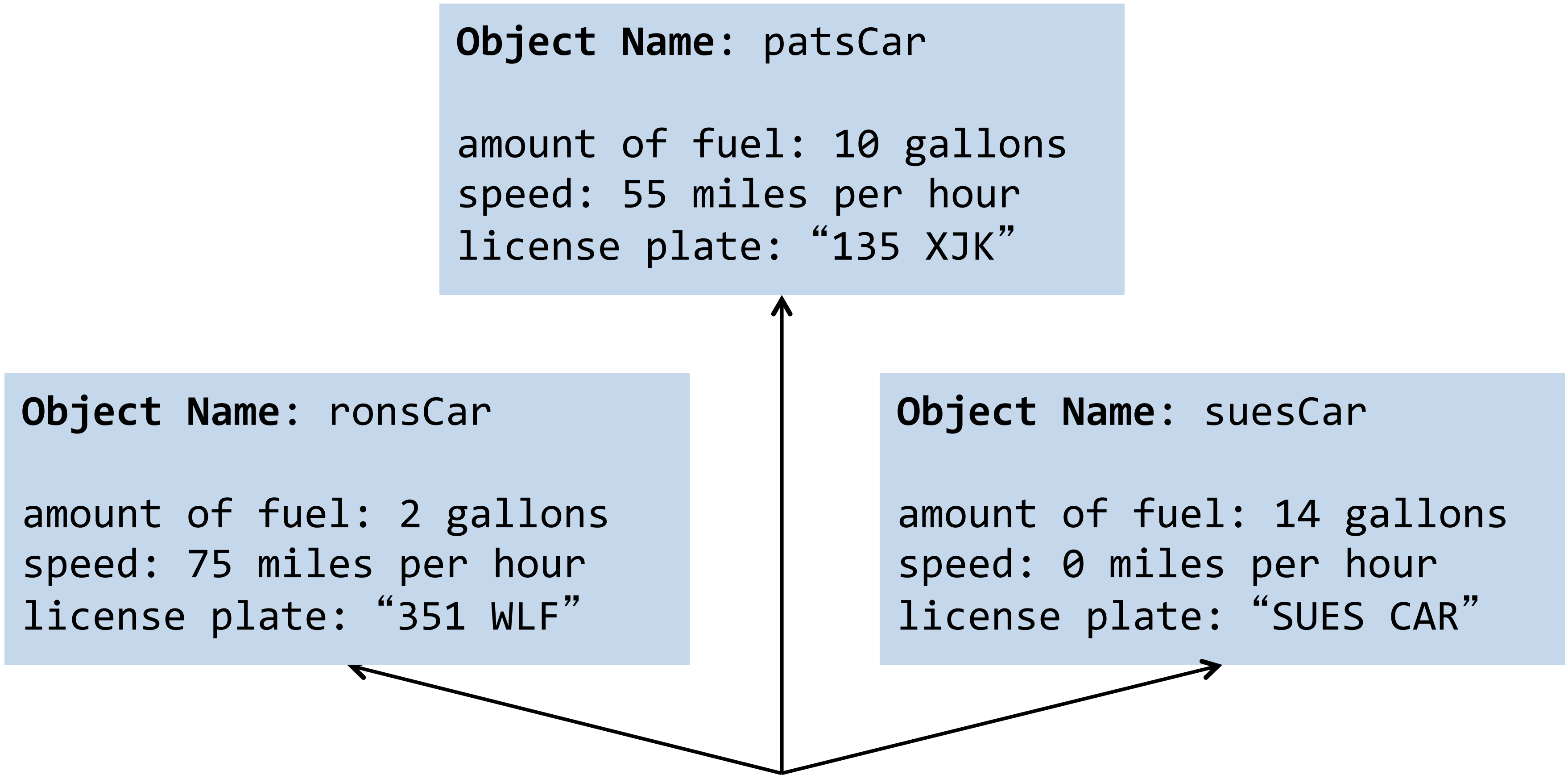
`accelerate:`

How: Press on gas pedal.

`decelerate:`

How: Press on brake pedal.

Objects, Instantiation



Object Name: patsCar

amount of fuel: 10 gallons
speed: 55 miles per hour
license plate: “135 XJK”

Object Name: ronsCar

amount of fuel: 2 gallons
speed: 75 miles per hour
license plate: “351 WLF”

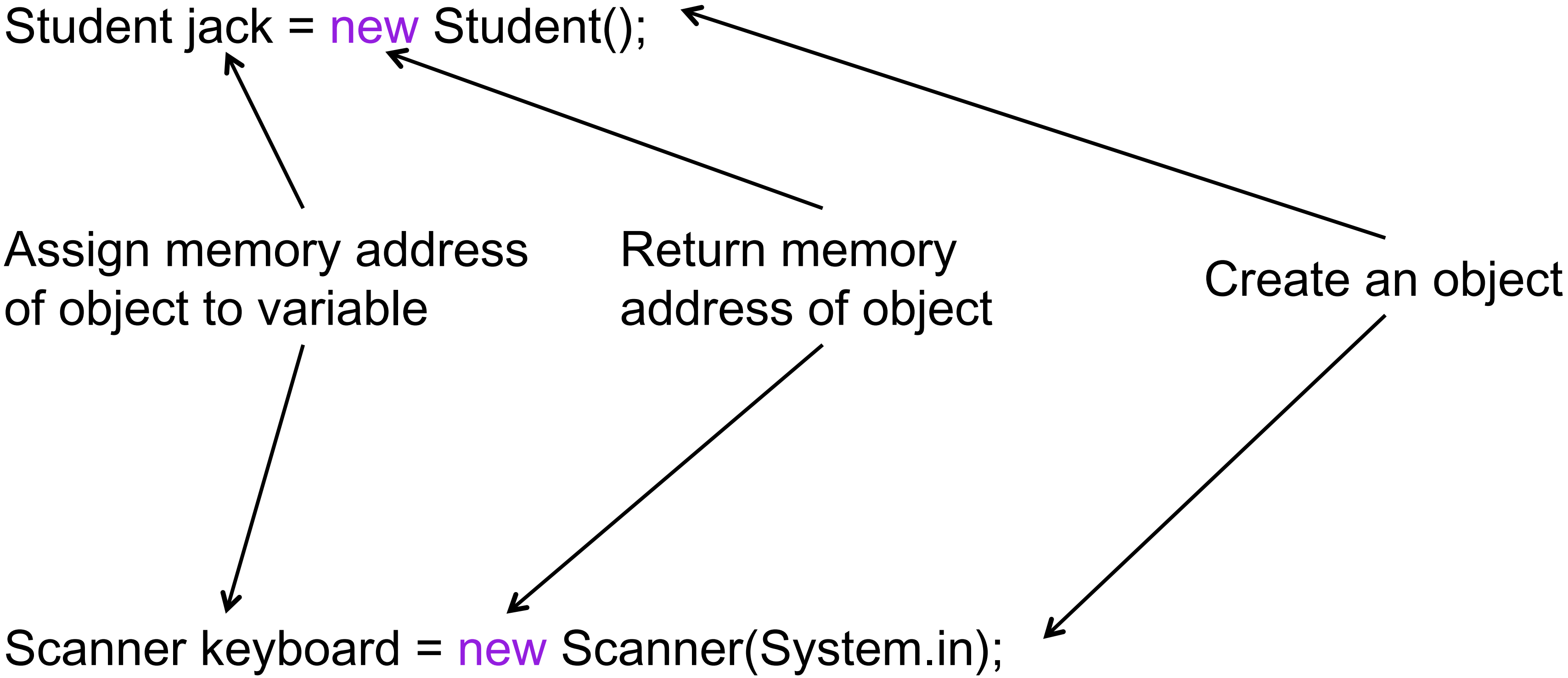
Object Name: suesCar

amount of fuel: 14 gallons
speed: 0 miles per hour
license plate: “SUES CAR”

Instantiations, or instances, of the class Automobile

Creating an Object

- Create an object *jack* of class *Student*



Student jack = new Student();

Assign memory address
of object to variable

Return memory
address of object

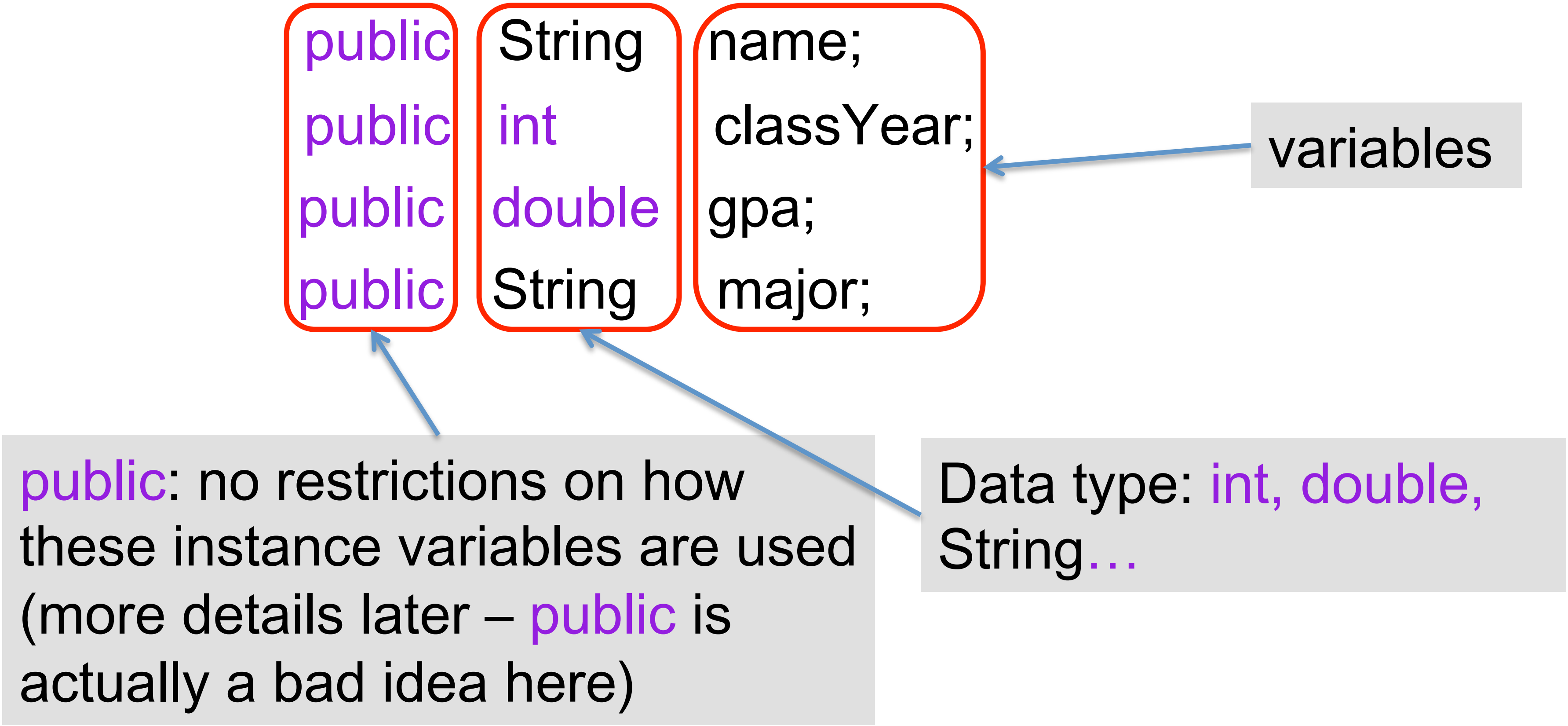
Create an object

Scanner keyboard = new Scanner(System.in);

- Create an object *keyboard* of class *Scanner*

Instance Variables

- Data defined in the class are called *instance variables*



```
public String name;
public int    classYear;
public double gpa;
public String major;
```

variables

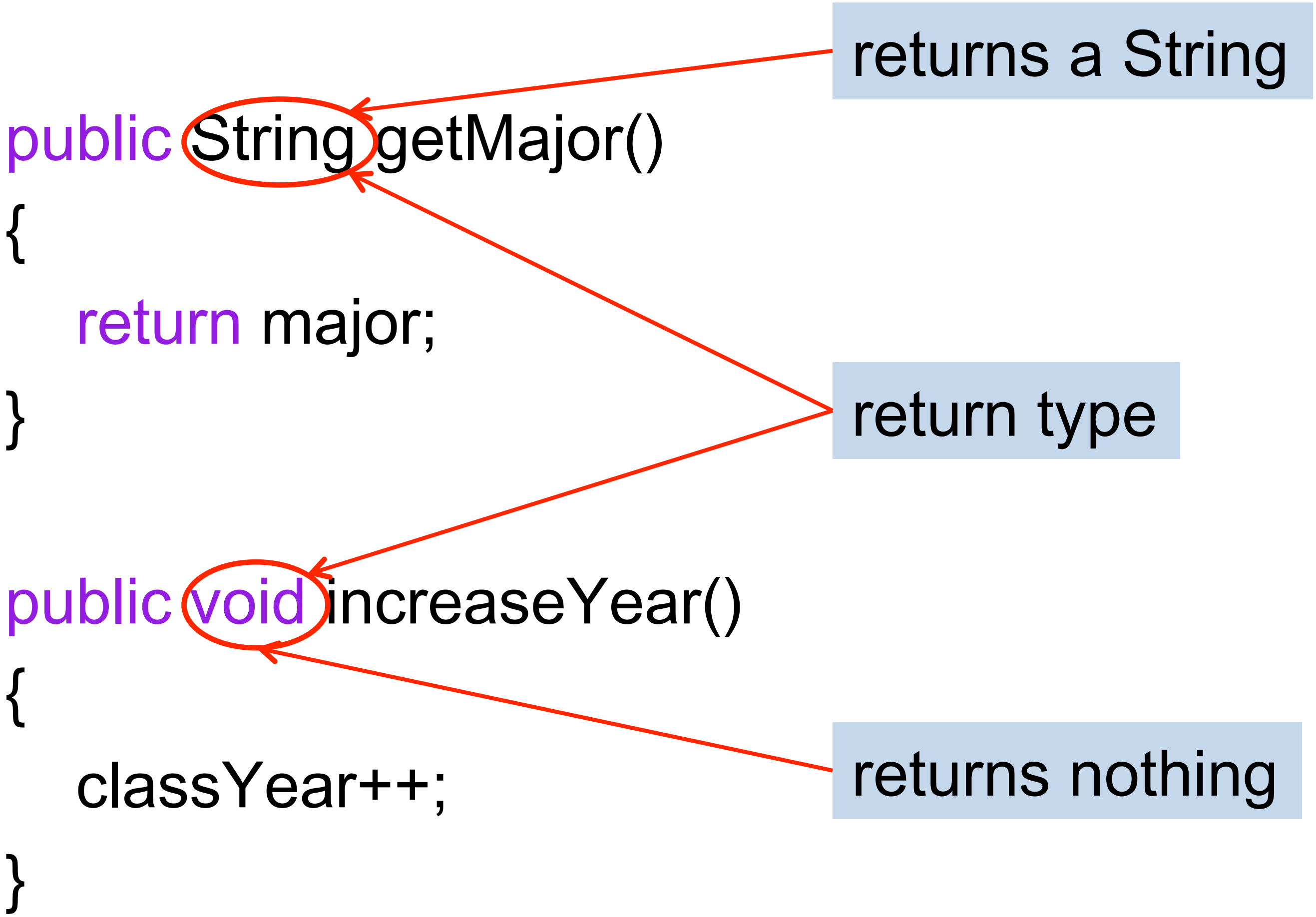
public: no restrictions on how these instance variables are used (more details later – public is actually a bad idea here)

Data type: int, double, String…

Methods

- Two kinds of methods
  - Methods that return a value
    - Examples: String’s *substring()* method, String’s *indexOf()* method, etc.
  - Methods that return nothing
    - Perform some action other than returning an item
    - Example: System.out.println()

Methods



```
public String getMajor()
{
    return major;
}

public void increaseYear()
{
    classYear++;
}
```

returns a String

return type

returns nothing

Calling Methods That Return Nothing

- Object, followed by dot, then method name, then ()
  - Order, type, and number of arguments must match parameters specified in method heading
- Use them as Java statements

```
Student jack = new Student();
jack.classYear = 1;

jack.increaseYear();

System.out.println(“Jack’s class year is ” + jack.classYear);
```

Calling Methods That Return a Value

- Object, followed by dot, then method name, then () (the same as before)
- Use them as a *value* of the type specified by the method’s return type

```
Student jack = new Student();
jack.major = “Computer Science”;

String major = jack.getMajor();

System.out.println(“Jack’s full name is ” + jack.getName());
System.out.println(“Jack’s major is ” + major);
```

Local / Instance Variables

- Instance variables
  - Declared in a class
  - Confined to the class
  - Can be used in any method in this class
- Local variables
  - Declared in a method
  - Confined to the method
  - Can only be used inside the method

```
public class Student
{
    public String name;
    public int classYear;
    public String major;

    public void printInfo(){
        String info = name + “:”
           + major + “:” + classYear;

        System.out.println(info);
    }

    public void increaseYear(int inc)
    {
        classYear += inc;
    }
}
```

An Example

```
public class Student
{
    public String name;
    public int classYear;
    public String major;

    public void printInfo()
    {
        String info = name + “: ” + major + “: ” + classYear ;
        System.out.println(info);
    }

    public void increaseYear(int inc)
    {
        classYear += inc;
    }
}
```

- *info* is a local variable declared inside method *printInfo()*
- can only be used inside method *printInfo()*

- *classYear* and *name* are instance variables
- can be used in any method in this class

An Example

```
public class Student
{
    public String name;
    public int classYear;
    public String major;

    public void printInfo()
    {
        String info = name + “: ” + major + “: ” + classYear ;
        System.out.println(info);
    }

    public void increaseYear(int inc)
    {
            classYear += inc;
            info = “info changed a bit”; }   ✗
}
```

- Java will not recognize *info*

Methods with Parameters

- Parameters are used to hold the value that you pass to the method
- Parameters can be used as (local) variables inside the method

```
public int square(int number)
{
   return number * number;
}
```

Parameters go inside parentheses of method header

Methods with Multiple Parameters

- Multiple parameters separated by commas

```
public double getTotal(double price, double tax)
{
    return price + price * tax;
}
```

Method Parameters and Arguments

```
public class SalesComputer
{
    public double getTotal(double price, double tax)
    {
        return price + price * tax;
    }
    // ...
}

SalesComputer sc = new SalesComputer();
double total = sc.getTotal( “19.99” , Color.RED);  ✗
double total = sc.getTotal(19.99);  ✗
double total = sc.getTotal(19.99, 0.065);  ✓
int price = 50;
total = sc.getTotal(price, 0.065); ✓
```

Automatic typecasting

Calling Methods from Methods

- In a method’s body, we can call another method
  - receiving_object.method();
- If calling a method in the same class, we do not need receiving_object:
  - method();
- Alternatively, use the this keyword
  - this.method();

Several Common Mistakes

- Unwanted semicolon after if / for statements

```
if (a>b); // this semicolon causes an empty if-branch
   c++; // this line is always executed
```

```
for(int i = 0; i<10; i++); // this semicolon indicates an empty loop body
   c++; // this is executed only once
```

- Unpaired brackets
  - Use indentation to help checking
  - Use Eclipse’s auto format function

Indentation

- Indentation
  - Makes code easier to read
  - Helps with finding syntax and logic errors
  - Indent code that goes between { and }

- Be consistent!

Next Class

- Go through questions from mid-term practice worksheet
- Q&A