MIDTERM REVIEW

http://pages.cpsc.ucalgary.ca/~tamj/233/exams/midtermInformation.htm

REMINDER

- Midterm
  - Time: 7:00pm - 8:15pm on Friday, Mar 1, 2013
  - Location: ST 148
  - Cover everything up to the last lecture before the reading week
  - Review:
    - Practice questions are available under the “Midterm” section on the course website
    - Review slides
    - Instructor’s office hour: 1pm - 3pm in MS lab

INTRODUCTION TO JAVA

Lecture 2 - 6 + sample code

Tutorial 2 - 4

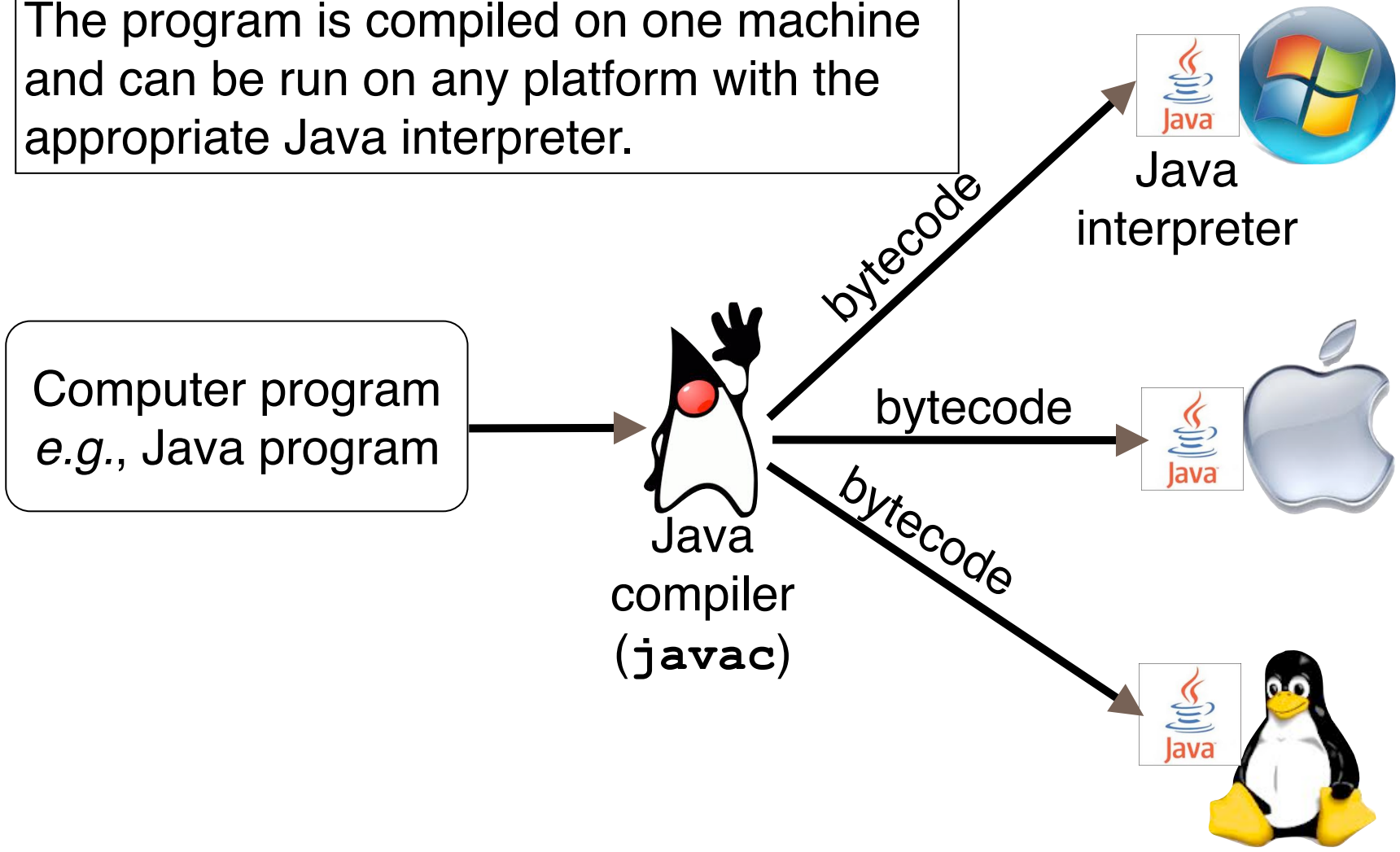
Quiz 1

Assignment 1

concept

JAVA

The program is compiled on one machine and can be run on any platform with the appropriate Java interpreter.



JAVA

- How does Java different from C++ and Python in terms of compilation and execution? (Hints: the two steps that you must perform to run your code.)

VARIABLE AND EXPRESSION

- What are the values of each variable after executing the following statements

```
double num = 1.005e2;
int x = 2;
float y = (int) num * (float) x;
int z = x++ * 3 - --x;
System.out.println("x = " + ++x);
System.out.println("y = " + y);
System.out.println("z = " + z--);
```

concept

SWITCH

- In C++/Java, the **`switch`** statement is an alternative to nested IF statement
  - It produces simpler code for certain case.
- Format:

```
switch (<name of the variable to be tested>)
{
   case <possible value>:
      body of this case
      break;
   case <possible value>:
      body of this case
      break;
   ...
   default:
      body of the default case
}
```

The variable can be of type **`byte`**, **`char`**, **`short`**, **`int`**, or **`long`**.

The value must match the type of the variable for a valid comparison.

The **break** is used to separate the cases.

SWITCH

- What are the values of each variable after executing the following statements

```
int code = 416;
switch (code)
{
    case 403:
        System.out.println("Calgary");
        break;
    case 708:
        System.out.println("Edmonton");
        break;
    case 416:
        System.out.println("Toronto");
        break;
    case 604:
        System.out.println("Vancouver");
        break;
    default:
        System.out.println("Unknown area code:" + code);
}
```

SWITCH

- What are the values of each variable after executing the following statements

```
int code = 416;
switch (code)
{
    case 403:
        System.out.println("Calgary");
    case 708:
        System.out.println("Edmonton");
    case 416:
        System.out.println("Toronto");
    case 604:
        System.out.println("Vancouver");
    default:
        System.out.println("Unknown area code:" + code);
}
```

SWITCH

- What are the values of each variable after executing the following statements

```
int code = 416;
switch (code)
{
    case 403:
        System.out.println("Calgary");
    case 708:
        System.out.println("Edmonton");
        break;
    case 416:
        System.out.println("Toronto");
    case 604:
        System.out.println("Vancouver");
        break;
    default:
        System.out.println("Unknown area code:" + code);
}
```

concept

"IF" STATEMENTS

- IF statement Format:

```
if (logical expression)
    body of 'if'
remainder of the program
```

- IF-ELSE statement Format:

```
if (logical expression)
    body of 'if'
else
    body of 'else'
remainder of the program
```

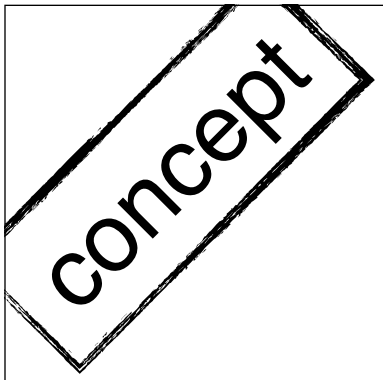
- IF-ELSE-IF statement Format:

```
if (logical expression)
    body of 'if'
else if (logical expression)
    body of 'else-if'
...
else if (logical expression)
    body of 'else-if'
else
    body of 'else'
remainder of the program
```

"IF" STATEMENTS

- Re-write the following statement using the "if-else-if" construct

```
int code = 416;
switch (code)
{
    case 403:
        System.out.println("Calgary");
    case 708:
        System.out.println("Edmonton");
        break;
    case 416:
        System.out.println("Toronto");
    case 604:
        System.out.println("Vancouver");
        break;
    default:
        System.out.println("Unknown area code:" + code);
}
```



LOOPS

- **`while`** loop format

```
initialization
while (logical expression)
{
    body
    update control
}
remainder of the program
```

- **`do-while`** loop format:

```
initialization
do
{
    body
    update control
}
while (logical expression);
remainder of the program
```

- **`for`** loop format:

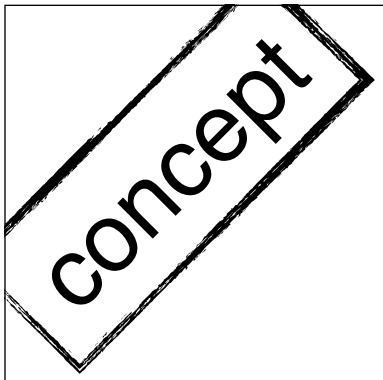
```
for (initialization; logical expression; update control)
{
    body
}
remainder of the program
```

LOOPS

- How many times the loop body will be executed?

```
int num = 100;
int total = 0;
int i = 0;
do
{
    total = total + i;
    i++;
} while (i <= num);
```

- Re-write the above code segment using the **`for`** loop construct.
- Re-write the above code segment using the **`while`** loop construct.



JAVA I/O: INPUT

- Input from command-line:

```
<type> <variable name> = args[0];
<type> <variable name> = args[1];
```

- User input:

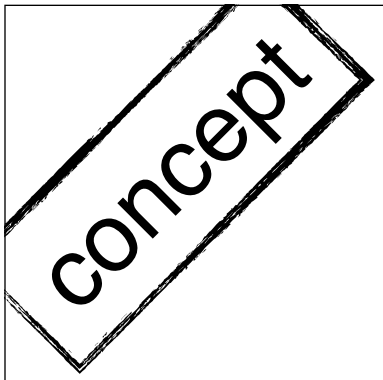
```
import java.util.Scanner;
...
Scanner <name of scanner> = new Scanner(System.in);
<type> <variable name> = <name of scanner>.<method>();
...
```

  - Useful **`Scanner`** methods:

    `nextInt(); nextLong(); nextFloat(); nextDouble(); nextLine();`

- Input from a file:

```
import java.io.*;

...

String <filename> = args[0];  // Getting the file name from the command line
FileReader <filereader> = new FileReader(<filename>);  // Open the file for read
BufferedReader <buffer> = new BufferedReader(<filereader>);  // Initialize the buffer
String line = null;

// Read each line of the file
do
{
    line = <buffer>.readLine();
    ...
} while (line != null);      // Checks for EOF
```



JAVA I/O: OUTPUT

- Format:

```
System.out.print(<string or variable name> + <string or variable name> + ..);
```

OR

```
System.out.println(<string or variable name> + <string or variable name> + ..);
```

- An escape character (backslash \) is use to indicate that the character is part of the string rather than part of the instruction.

Escape	Meaning
\\	Backslash (keeps \)
\"	Double quote (keeps ")
\n	Newline, end of line
\t	Horizontal tab
...	...

- Format Output

```
// formatting integer output
System.out.printf/format("%<field width>d", <int variable>);
// formatting string output
System.out.printf/format("%<field width>s", <String variable>);
// formatting floating point output
System.out.printf/format("%<field width>.<precision>f", <float/double variable>);
```

JAVA I/O

- Try to print:
  `I said,`
  `"Don't do it"`

OOP: Classes and Info Encapsulation

Lecture 7 - 14 + sample code

Tutorial 5 - 7

Quiz 2

Assignment 2

concept

CLASSES

- A *class* is an abstract data type (ADT) that consists of attributes (fields) and methods (functions) that operate on the data.
  - Format:

```
public class <name of the class>
{
      instance fields/attributes (fields)
      instance methods
}
```

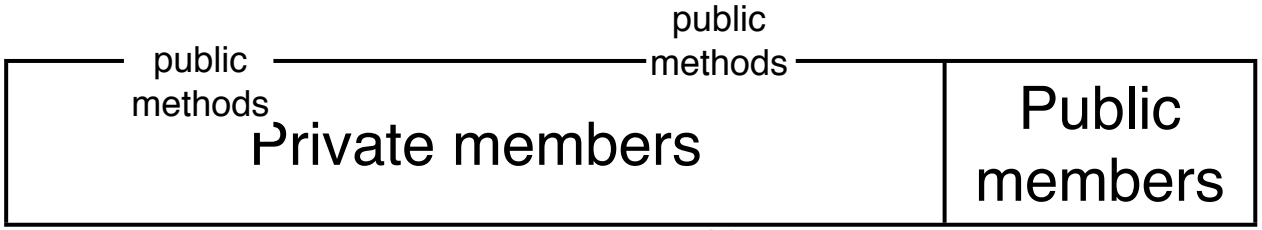
- An *object (instance)* is a variable of a class, and the declaration is called *instantiation*.
  - Format:

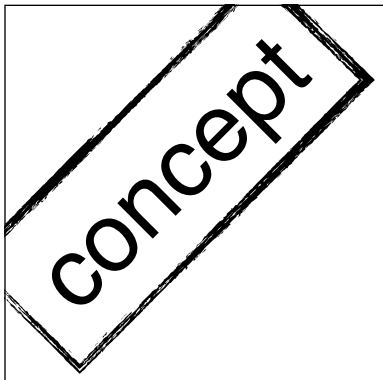
```
<class name> <object name> = new <class name> ();
```

- Encapsulation: information hiding

Accessor methods (*e.g.*, **`get`** method)

Mutator methods (*e.g.*, **`set`** method)





MEMBERS

- Members:
  - Data members: `<access modifier> <type> <name>;`
  - Methods:

`public` or `private`, but typically `private`

```
<access modifier> <return type> <method name> (<type> <param1>, <type> <param2>, ...)
{
    <Body of the method>
}
```

- Accessing members
  - A private members can only accessed by class methods (within the class)
    - Accessing data members: `this.<name>`
    - Access methods: `this.<method name> (arg1, arg2, ...);`
  - A public data member can be accessed by the object and class methods (both within and outside the class)
    - Accessing data members:

```
<object name>.<name>   // outside the class
this.<name>            // within the class
```

    - Access methods:

```
<object name>.<method name> (arg1, arg2, ...);   // outside the class
this.<method name> (arg1, arg2, ...);            // within the class
```

CLASSES

- Complete the code as instructed by the comments in **`formatName()`** and **`main()`**.

```
public class MyFormatter {
    private String first;
    private String last;
    private Scanner in;
    public MyFormatter () {
         first = null;
         last = null;
         in = new Scanner(System.in);
    }

    public void promptNames () {
         System.out.print("Enter first name: ");
         first = in.nextLine();
         System.out.print("Enter last name: ");
         last = in.nextLine();
    }

    public String formatName () {
         String name;
         // Call promptNames()

         // Name format: <last><tab><first>


         return name;
    }
}
```

```
public class Driver {
    public static void main (String [] args) {
         MyFormatter aFormatter = new MyFormatter();
         String formatted;
         // call formatName() and store results in "formatted"

        System.out.println(formatted);
    }
}
```

INFORMATION HIDING

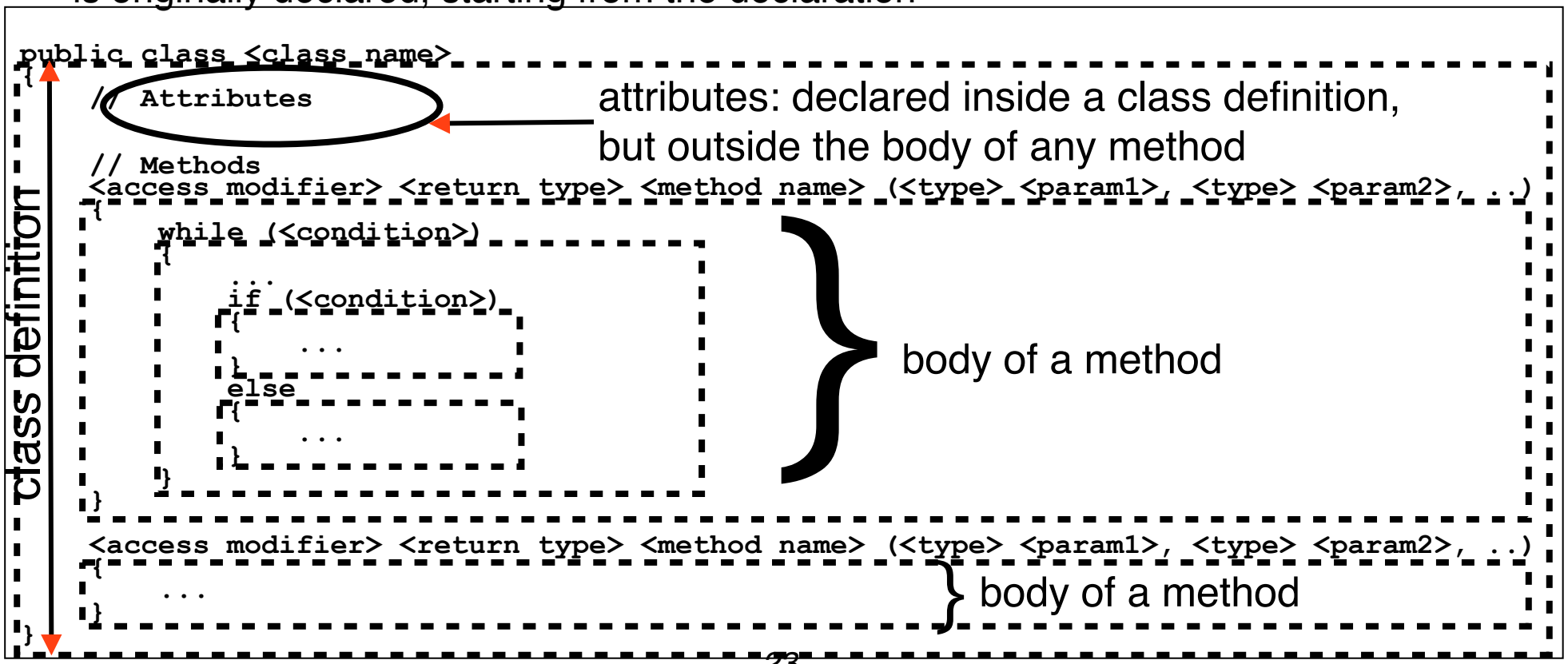
- Write an accessor/get and mutator/set method for the following class definition.

```
public class Student
{
    private int identificationNumber;

     public Student ()
     {
         identificationNumber = -1;
     }

      // Accessor




      // Mutator



}
```

concept

SCOPE

- Attributes: variables or constants declared *inside* the body of a class definition, but *outside* the body of any class method
- Local variables: variables or constants declared *within* the body of a class method
- In general, the scope of a variable, constant, or method is the inner-most block where it is originally declared, starting from the declaration



SCOPE OF ATTRIBUTES AND LOCAL VARIABLES

- Identify the body of a class definition, the body of a method, attributes, and local variables
- For each attribute and local variable, identify its scope.

```
1     public class Car
2     {
3         public String model;
4         private int odometer;
5         public void addToOdometer(int city, int road)
6         {
7             odometer += city + road;
8         }
9         private String plate;
10        public void setPlate(String plate)
11        {
12            plate = plate;
13        }
14    }
```

concept

CONSTRUCTOR & METHOD OVERLOAD

- A constructor is a special method that:
  - has the same name as the class itself
  - is invoked when creating the object
    (*e.g.*, `Student alice = new Student();`)
  - can be redefined by the programmers
    (e.g., you can specify how you want to initialize each data member)

default constructor
(provided by the language
by default)

- Format:

```
public <class name> (<type> <param1>, <type> <param2>, ...)
{
    // body of the constructor
}
```

- Overload methods are the methods with the same method name, but different method signature (the list of parameters)
- Commonly shared by methods with similar implementation for different tasks

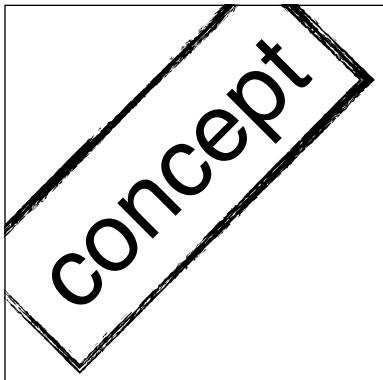
METHOD OVERLOAD

- Assuming the following methods are already included in the same class definition:

```
public double method()
public void method(int)
```

  - Identify invalid method overload:

```
public void method()
public int method()
private void method()
public int method(int, int)
public int method(int)
```



SELF REFERENCE

- For every (non-static) method of an object, there exists a reference to the object itself (keyword **`this`**)
- **`this()`**: invoke the constructor of a class
- It is usually used when overloading the constructor.
- Example:

```
public Student ()
{
    firstName = "";
    this.firstName = "";
    lastName = "";
    studentID = 0;
    grade1 = 0.0;
    grade2 = 0.0;
}

public Student (String firstName, String last)
{
    this();
    firstName = firstName;
    this.firstName = firstName;
    lastName = last;
}
public Student (String first, String last, int id)
{
    this(first, last);
    studentID = id;
}
```

equivalent

not the same (shadowing)

SELF REFERENCE & METHOD OVERLOAD

- What is the output of the following program?

```
public class Tracer {
    private int x;
    private int y;

    public Tracer() {
        x = 7;
        y = 13;
    }

    public Tracer(int x, int y) {
        this();
        this.x = x;
        y = y * 2;
    }

    public void display() {
        System.out.println(x + " " + y);
    }

    public void method() {
        int x = 2;
        x = x * 10;
        y = x + y;
    }

    public void method(int a) {
        x = a;
        y = this.y * 2;
    }
}
```

```
public class Driver {
    public static void main(String [] args)
{
        Tracer aTracer = new Tracer();
        aTracer.display();

        aTracer = new Tracer(888,666);
        aTracer.display();

        aTracer.method();
        aTracer.display();

        aTracer.method(707);
        aTracer.display();
    }
}
```

ARRAYS

Lectures 15 - 17 + sample code

Tutorials 8

concept

OPERATIONS: CREATE

- Declaration: creating only a reference to the array.
  - Format: `<type>[] <array name>;`
- Allocation: allocate a contiguous memory space
  - Format: `<array name> = new <type> [<# of elements>];`
- Access: an array is really a list of indexed elements
  - Format: `<array name> [index]`
- Modify: modifying array elements
  - Format: `<array name> [index] = ...;`
- Common operations: initialization, adding, removing, searching, and displaying
  - They all share the general loop construct

```
<type>[] <array name> = new <type> [<# of elements>];
int next = 0;  // to keep track of the next free space in the array
...
for (int i = 0; i < arr.length; i++)
{
    // accessing or modifying individual elements
    // and algorithms for different operations.
}
```

or "next", depending on the algorithm

```
public class Manager {
    public final static int MAX = 10;
    public static final int EMPTY = -1;
    private int [] salaries;
    private int lastElement;      // Note this is the lastElement, not "next"

    public Manager () {
        lastElement = EMPTY;
        salaries = new int [MAX];
        for (int i = 0; i < MAX; i++)
            salaries[i] = EMPTY;
    }

    public boolean isEmpty () {
        if (lastElement <= EMPTY)
            return true;
        else
            return false;
    }

    public boolean isFull () {
        if ((lastElement+1) == MAX)
            return true;
        else
            return false;
    }

    public void display () {
        for (int i = 0; i <= lastElement; i++)
            System.out.println(salaries[i]);
        System.out.println("");
    }

    // Adds new elements to the end of the list.
    public void add (int aSalary) {
        lastElement++;
        if (isFull() == false)
            salaries[lastElement] = aSalary;
        else
            System.out.println("Can't add: list is already full");
    }

    // Returns index of first matching salary or EMPTY if no matches.
    private int indexAt (int aSalary) {
        ...

        return index;
    }

    // Use indexAt() to find the matching salary, and then remove it
    public void remove (int aSalary) {
        ...
    }
```

concept

ARRAYS OF OBJECTS

- Declaration + allocation format:

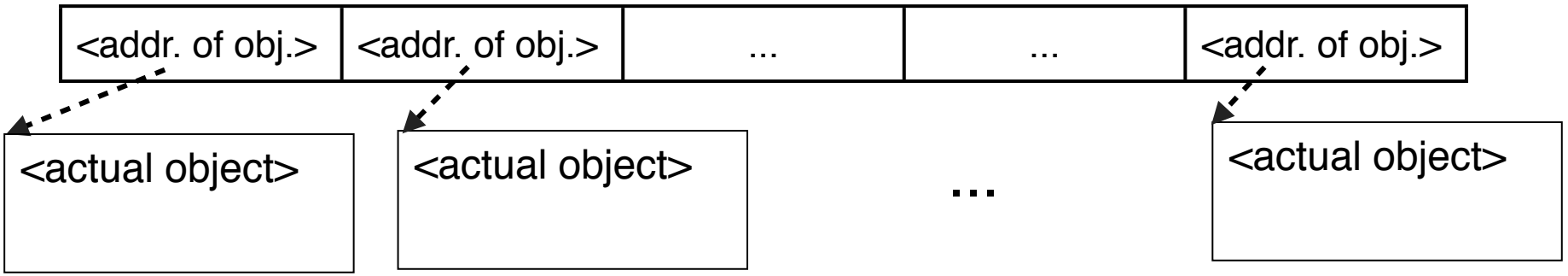
```
<class name>[] <array name> = new <class name>[# of objects];
```

  - Each element in the array is a reference. By default, all references are set to null.
- Access an array element:

```
<array name>[index]
```

- Initialization format:

```
for (int i = 0; i < <array name>.length; i++)
{
    <array name>[i] = new <class name>(...);
}
```



ARRAYS OF OBJECTS

- Modify the code in `main()` so that an array of `SIZE Car` objects is created.
- Set the price of each car as follows: 10000, 20000, 30000, ... (the price is increased by 10000 from one car to the next in the array)
- Display the attributes of each car object, one car per line with the format:
  <model information> <price>

```
public class Car
{
    private String model;
    private int price;

    public Car ()
    {
        setModel("No name");
        setPrice(-1);
    }

    public String getModel ()
    {
        return model;
    }

    public int getPrice ()
    {
        return price;
    }

    public void setModel (String aModel)
    {
        model = aModel;
    }

    public void setPrice (int aPrice)
    {
        price = aPrice;
    }
}
```

```
public class Start
{
    public static void main(String [] args)
    {
        final int SIZE = 10;
        int i;
        Car[] myCollections;









        }
    }
}
```