CS102

Introduction to data structures, algorithms, and object-oriented programming

Java Statements

Statements:

a. package inclusion,
b. import existing packages,
c. variable declarations, initializations, and instantiations
d. assignments (z = y),
e. conditionals (if, else), switch
f. loops (for, while, do-while)
g. flow of control modifiers (break, continue),
h. method calls,
i. method returns.

Items a and b occur before class definition. Items c through i can occur only within class {}s

Structure of a Java program

```
package example;

import java.util.*;

public class Example{

   <global variable/constant definitions>

   <method definitions> (){
        <local variable definitions>
   }
   <etc…>
}
```

package statement joins classes in a single application. Not required.

import statements make use of Java libraries easier

Java Packages

A Java package is a compressed collection of classes. You need to know the package a class is written in before you can use the class.

So how do you find the package that contains a particular class, e.g. String, Color, Scanner?

You could spend hours going through the java API, but an easier way to find the documentation is searching for the class name in a browser: Scanner java API
Choose link from docs.oracle.com

You can even look up methods this way:
parseInt java API

Java Packages – java.lang

There is a package called java.lang that you never have to import because it is imported automatically.

This package provides fundamental classes in the Java programming language.

java.lang contains many of the most commonly used classes such as System, String, Object (the root of the class hierarchy), and many more.

Data Types

Data types come in two main categories:

primitive types: part of the formal syntax of Java:
- they evaluate only to themselves (literals)
- they each have a set of operations that can be used on variables and literals of the type

reference types: created from classes written by you and those that are available in Java libraries:
- data of reference type are known as objects
- objects are created using the keyword "new"
- objects have instance variables (state) and instance methods (behaviors)

Java Naming Conventions

Classes, variables, methods, and constants are each identifiers named according to programmer conventions:

- Class names should start with a capital letter and all multiple-word class names should start each word with a capital letter.
- Variable and method names start with lowercase and if they have multiple words, the start of each word except the first should be capitalized.
- Constant names should be written in all capital letters, with words separated by _ (underscore).
- Keywords and package names are all lowercase.

Java Naming Rules

1. Names can contain alpha characters, numbers, $, and _
2. Names cannot start with a number.

The compiler checks for violations of naming rules.

Primitive Data Types

A data type is a set of literal values and a set of operations on those values.

There are four primitive data types that can be considered the basis of the Java language:

1. Integers, with arithmetic operators (**int**)
2. Real numbers, with arithmetic operators (**double**)
3. Booleans, with logical operators (**boolean**)
4. Characters, alphanumeric symbols inside 's (**char**)

Strings, anything written inside "s (**String**)

Strings are not actually a primitive type, but they are used like a primitive type. String is a class, with its own methods.

Wrapper Classes

Each primitive data type has an associated reference type defined in the java libraries. These classes provide methods to convert primitives into objects and objects into primitives.

The wrapper classes include:

1. Integer
2. Double
3. Boolean
4. Character

Primitive Types in Java

Type	Memory	Smallest	Largest
`byte`	8 Bits	-128	127
`short`	16 Bits	-32768	32767
`int`	32 Bits	-2147483648	2147483647
`long`	64 Bits	$\approx -9.2*10^{18}$	$\approx 9.2*10^{18}$
`float`	32 Bits	$\approx \pm 1.4*10^{-45}$	$\approx \pm 3.4*10^{38}$
`double`	64 Bits	$\approx \pm 4.9*10^{-324}$	$\approx \pm 1.8*10^{308}$
`char`	16 Bits	NA	NA
`boolean`	8 Bits	NA	NA

Reference or Object Types

Reference types are created by declaring a variable with a class name. For example:

```
import java.awt.*; // package needed for Color class

public class Man {
   // declaration and initialization of instance variables
   static Color skinColor = Color.BLUE; // class variable
   static Color shoeColor = Color.BLACK; // class variable

   Color shirtColor, pantsColor; // declaration of instance variables
   double shoeSize;  }
```

To create an object of type Man, either in the main method of this class or from another class, use this syntax:

```
Man perry = new Man();
```

Purpose of a Java program

A Java program (i.e., class) is either:

1. a library of *static* methods (functions) that may return values or just have side effects (like Scheme); or

2. a data type definition: a template for creation of objects.

There is one static method that *must* be included in every Java application: the **main** method. Each application starts execution at a method with the following signature:

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

Reference or Object Types

Reference types can contain data stored in fields and method definitions that can be called on objects of the type.

```
import java.awt.*; // Needed for Color class

public class Man {
   static Color skinColor = Color.BLUE; // class variable
   static Color shoeColor = Color.BLACK; // class variable

   Color shirtColor, pantsColor; //  object instance variables
   double shoeSize; // primitive instance variable

    public void setColors(Color shirt, Color pants, double size) {
    // instance method
          shirtColor = shirt;
          pantsColor = pants;
          shoeSize = size;
    }}
```

```
Definition of instance
method returning void. This
method sets the values of 3
instance variables
```

Variable declaration

Java is a *strongly typed language*, meaning that all variables must be declared as a particular type before they can be used in an expression.

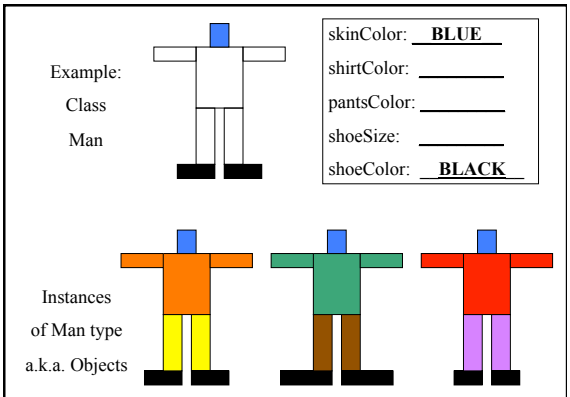
Variables declared inside a method block are called "local variables". Those outside a method block are "global variables".

To use the integer variable num as either a global or local variable, the use must be preceded by the declaration

```
int   num;     // declares num as integer
num = 15;    // initializes value of num
```

or

```
int num = 15; // all on one line
```



Method signatures

The first line of a method is called the signature. The signature tells the programmer what they need to know to use the method, including a list of comma separated type name pair parameters in parenthesis. Methods always contain code inside a set of {}s.

Form of main method signature:

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

The main method is the single starting point of execution; other methods must be called on some trail starting from the main method.

Static Methods and Fields

Examples of static method signatures:

```
public static int getNumTicketsSold(TicketBox tb)
public static boolean spellCheck(String word)
public static void printVars(int a, double b, String c)
```

Global variables (fields) can also be declared static:

```
public static Color skinColor = Color.BLUE;
public static Color shoeColor = Color.BLACK;
```

Static methods and fields belong to the entire class and are accessed through the class name when used by other classes. Static fields are the same in all objects made from the class.

Non-Static Methods and Fields

Forms of non-static method signatures:

```
public int getAge(Man guy)
public boolean spellCheck(String word)
public void printVars()
```

Global variables (fields) can also be declared non-static, in which case, each object can have unique values for those fields:

```
public Color shirtColor;
public Color pantsColor;
public int shoeSize;
```

Lack of keyword "static" means method/field belongs to an object created from the class by using the keyword "new".

Java Keywords (47)

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

Commonly Used Keywords (40)

```
abstract    double      int         super
assert      else        interface   switch
boolean     enum        long
break       extends                 this
byte        final       new         throw
                        package     throws
char        float       private
catch       for         protected   try
class       if          public      void
continue    implements  return
            import      short       while
do          instanceof  static
```

REALLY COMMONLY USED Keywords (30)

```
            double      int         super
            else        interface   switch
boolean
break       extends                 this
            final       new         throw
                        package     throws
char                    private
catch       for                     try
class       if          public      void
            implements  return
            import                  while
            instanceof  static
```

Object Declaration statements

Objects of class type must also be declared before they are instantiated (can be on the same line)

```
Man bobby = new Man();
```

`new` is keyword that calls a special part of each class called the *constructor*.

The main purpose of a constructor is to set values of instance variables (but they can have other executable code inside too). Constructors always have the same name as the class they are written in and they have no return value.

Static Variable Declaration

Variables can also be declared inside the class braces, but outside any methods.

These are "global" or "class variables", accessible in any method of the class.

Class variables are defined for the entire class.

Class variable declaration can specify a variable is static (the same for all objects of the class)

static int i = 12;

Assignment Statements for primitives use =

A primitive variable name is actually a memory location. The value you assign to that name becomes the content of that memory location.

For example, after declaring the integer i, you could assign a value (sometimes a literal, sometimes an expression) to i as follows: **i = 5;**

You can also combine declaration and initialization on one line as follows:

String name = "Nancy";

Literals

Numeric literals are discussed in Eck, Sect 2.2.3. We will use primarily integers and doubles.

Character literals are written between apostrophes:
'A' , 'b', '\n' etc.

String literals are character enclosed in "'s.

true and false are boolean literals.

Method calls

Calling a function. A method *call* always has the form:

methodName(comma-separated argument list);

The line
System.out.println("Hello World");
// is a call to the println method of System.out

For example, let's look at a Java program.

Non-void Static Methods

Methods that return a non-void type must contain 1 or more return statements:

```
public class MethodExample{

    public static void main (String[] args) {
        int num = 17;
        System.out.println("num cubed is " + cube(num));
    }

    public static int cube(int n) {

        return (n * n * n);
    }
}
```

call of cube method passing in integer

definition of static method returning an integer

Notice that the cube method is declared static, like main. This means that the cube method is a *class* method. It must be called on the class name using the . ("dot" or selection) operator: MethodExample.cube(num).

In the same class, there is no need for the class name before the method call

Non-void, Non-static Methods

In order for the static method main to call a non-static (instance) method, it must create an object of the class type to call the method on.

```
public class MethodExample{

    public static void main (String[] args) {
        int num = 17;
        MethodExample methodCube = new MethodExample();
        System.out.println("num cubed is " + methodCube.cube(num));
    }

    public int cube(int n) {

        return (n * n * n);
    }
}
```

declaration and instantiation of MethodExample object

call of cube method passing in integer

definition of instance method returning an integer

Notice that the cube method is not declared static. This means that the cube method is an *instance* method, which means that it must be called on an *object* of type MethodExample.