Java™ Basics and Object-based Programming

Part 1, Csc 115 Fall 2002

Dr. Storey

Reading assignment

- Chapter 1 in textbook
- Study Java libraries extensively
  - http://java.sun.com/j2se/1.3/docs/api/overview-summary.html
  - java.lang
    - Boolean, Integer
    - Math (PI, max, min, sin, cos, random(), round(), sqrt())
    - Object (clone(), equals())
    - String (CharAt(), CompareTo(), equals(), length())
    - System (println(), print(), flush(), Assignment 1)
  - java.io
    - BufferedReader (Section 1.6 in textbook)
    - Stdin, flush(), readLine()
  - java.util
    - List, LinkedList, Iterator
    - Observer
    - Calendar, set(), get() (Assignment 1)
    - Hashtable
    - Random (Assignment 1)
    - Stack
  - The more you know what is in these libraries, the less code you have to write.

Topics to be covered….

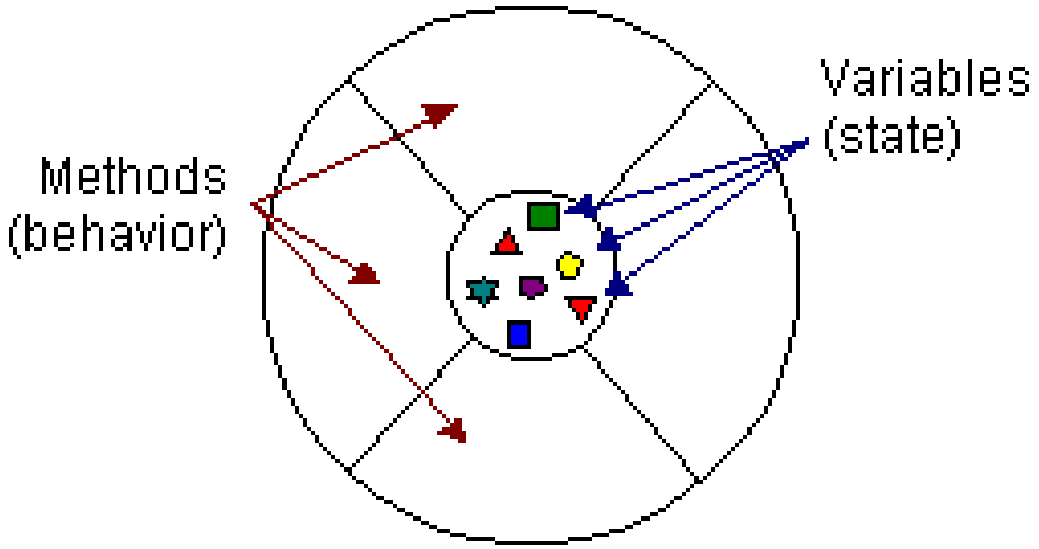
- Today
  - Classes and Objects
  - Methods
  - Primitive Types
  - Variables
- Next class or two
  - References
  - Parameter passing
  - Arrays
  - Control Flow
  - Input and Output
  - Strings
  - Expressions, operators
- Followed by….
  - Packages
  - Castings
  - Inheritance
  - Interfaces
  - Modifiers
  - Static members of a class
  - Exception

What is an object?

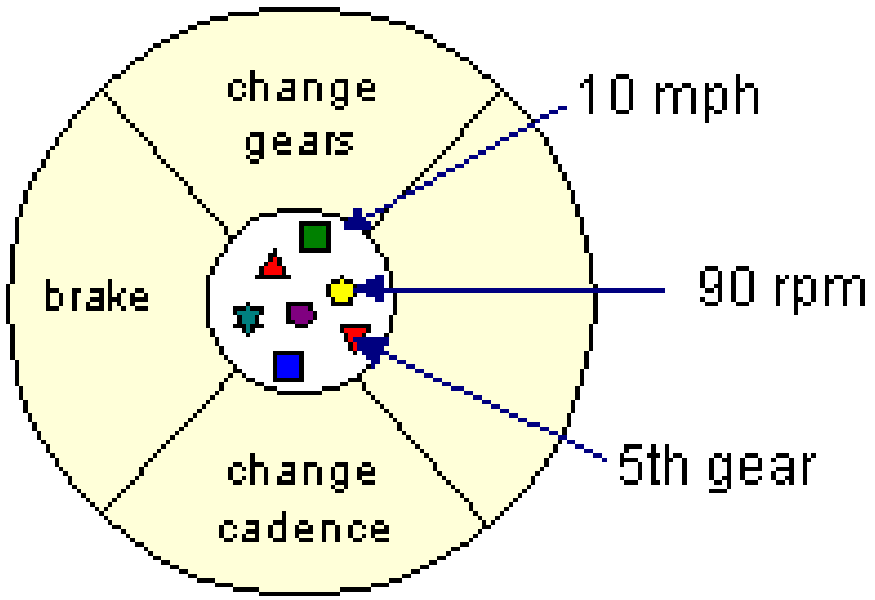
- The main “actors” in an OO programming language are *objects*
  - ✍ Objects are alive ✍
    - They can represent real world objects (such as dogs, bicycles) or abstract concepts (such as a GUI event)
  - ✍ Objects have state and behaviour
    - State determines everything an object knows
    - Behaviour determines all of the actions an object can do

Definition: An object is a software bundle of variables and related methods.

One view of an object…



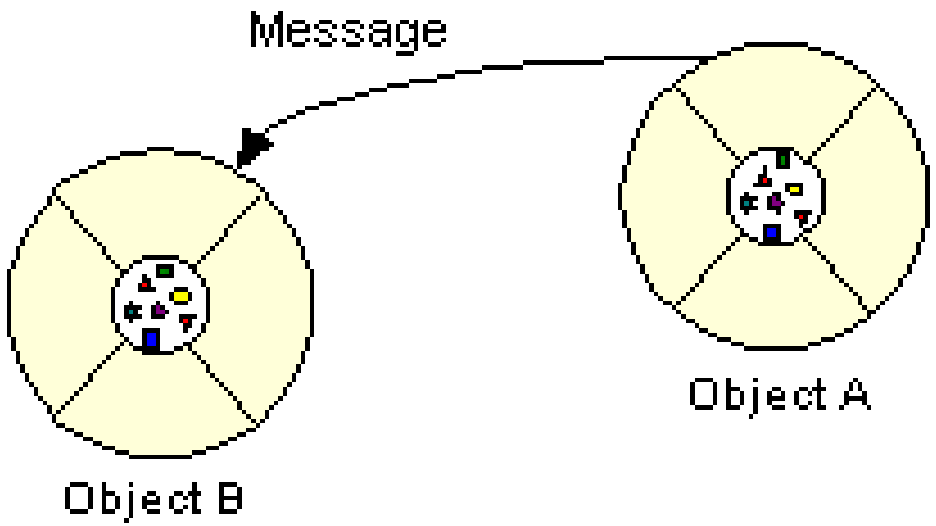
An example of an object



Why objects?

- They provide *encapsulation* of its methods and variables
- Lends to more *modular* code (information hiding)

Objects communicate via messages



An object's behavior is expressed through its methods therefore message passing supports interactions between objects

Objects don’t have to be part of the same program or even on the same machine to send messages to each other

What is a class?

- We often have objects of the same kind (type)
- Using object-oriented terminology, we say that a particular dog object is an *instance* of the class of objects known as dogs.
- Dogs have some state in common (number of legs etc) and behavior (barking ability) in common. However, each dog's state is independent of and can be different from that of other dogs.
- We can take advantage of the fact that objects of the same kind are similar and create a blueprint for those objects. A software blueprint for objects is called a *class*

Definition: A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.

Classes and objects

- Every object is an instance of a *class*

- A class consists of members
  - There are two categories of class members
    - Fields or variables
    - Methods
  - Methods of an object operate (i.e., access, modify) on its fields

  - A class defines *types* for all of its fields and variables
  - The type of a field can be a *primitive* type or *reference* to a user defined type

Class members

- Fields
  - Data associated with an object
  - Represent and store the *state* of an object
  - The type of an field or a parameter can be primitive or reference
  - All fields are initialized to default values automatically
  - Examples

```
public int k = 17;
public Point p = new Point(17,12);
```

- Methods
  - Define the behaviour of the objects instantiated from that class
  - Methods are also called functions or procedures (no return types)

```
void doNothing() { }
ComplexNumber makeComplex(double r, double i){ /* … */
int findSock(Color c, Socks[] a) { /* … */ }
double[] getGrades() { /* … */ }
```

Our first piece of code…

- Let's look at how to write a very class in Java….
- An Interactive programming session: FirstClass.java

- Recap:
  - FirstClass – defines blueprint for a set of objects of type FirstClass
  - Let's add another instance…. Now we have two instances (objects) of type FirstClass

Class declaration

Syntax

```
[modifiers] class ClassName [extends SuperClassName]
  [implements Interface1, Interface2, …] {
   class member declarations;
}
```

Legend:

[ ] – optional

Bold – keywords

Note: identifiers (must begin with a letter or any other unicode character)

Accessing members

- Dot notation
- Accessing instance members

```
objectName.classMember
objectName.field
objectName.method()
```

Methods

- *Every* method in Java has to be specified in the body of some class
- 2 parts:
  - **Signature:** defines the name and parameters (note, not the return type)
  - **Body**: what the method actually does for a living

- Syntax:

```
[<method-modifiers>] <return_type> <method_name>
                        ([param_type param1, ….] {
            // method body
}
```

- Note: Use the keyword **void** if there are no return types

Constructors

- A special type of method
- Instantiates and initializes objects
- Has same name as the class
- A class can have many constructors; all have the same name, but all signatures must be different
- A **public**, no-argument constructor is provided by the Java run-time environment if the class does not define one
- Can only be called using new

Syntax:

```
[construct_modifiers] <constructor_name> ([<param_list>) {
        // constructor body
}
```

Notice anything different from other methods?

main() method

- The main entry point of a Java program
- This is the first routine called by the operating system
- Specific signature:

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

- Each class can have a `main()` routine for testing purposes

Statement blocks and local variables

- A statement block is in between { }
- Method bodies and statement blocks can have statement blocks nested within them
- Local variables (either a base type or reference to an instance of some class) can be used within statement blocks

Primitive types

- Primitive types are defined by the language:
  - `byte, short, int, long, float, double, boolean, char`

- All primitive types have literals
  - A literal is an unnamed constant value
  - Examples

int	42	052	0x2a	
double	42.0	42.	4.2e1	42d
float	42.0f	.42e2f		
boolean	true	false		
char	'c'	'\n'	'\\'	'\"

- You can *wrap* primitive data inside objects, if necessary
- Sometimes useful to treat all variables uniformly

```
Integer intWrapper = new Integer(3);
int i = intWrapper.intValue();
```

Storage allocation for variables

- Allocating an object

```
String name;
Abc k;
```

- Allocating a cell for a variable of a primitive type

```
int j;
double d;
```

- Instantiation and initialization

```
j = 3;
d = 3.14159;
name = “Bette”;
k = new Abc();
```

- Variables of primitive types always given an initial value whenever an object containing them is created (O for all, except boolean which is set to false)

Variables Quiz

```
int sumSquares(int n) {

   partialSum = 0;

   int i;

   while (i <= n) {
      int square = i*i;
      partialSum += square;
      i++;
   }

   System.out.println(”last square = ” + square);

   return partialSum;
}
```

Identifiers and Reserved Words

- Identifiers are used as names for variables, constants, classes, methods, etc.

- Must follow certain rules:
  - must begin with a letter
  - may contain additional letters and digits
  - all characters are significant
  - case sensitive
  - must not conflict with a reserved word

Identifier Quiz

Identifier	Valid?
sum	
4you	
salary%	
MEDIUM	
long	
longint	
Double	
NO_VALUE	
_12	
goto	
Rect$1	
? ? ? ?	

Naming conventions

- Variables, fields, parameters
  - ✍ Mixed case, start with lower case
    - k
    - inputMode
- Classes, constructors
  - ✍ Mixed case, start with upper case
    - Person
    - Clock
- Constants
  - ✍ All upper case
  - ✍ PI, MAXNUMBER, LASTINDEX
- Methods
  - ✍ Mixed case, start with lower case, parenthesis
    - getAge()
    - setUserID()
- Packages
  - ✍ All lower case
    - awt
    - swingx
    - project