Modular Software Development

Department of Computer Science and Engineering

University of Moratuwa

Declarations and Types

- Variables must be declared before being used
  - A declaration includes the type of the variable
  - Two kinds of types, **primitive** and **object**
  - Primitive types include
    - int
    - Long
    - boolean (true or false)
    - double (floating-point numbers)
    - char (characters) – Object types include
  - Object type include
    - String (a sequence of characters, i.e. text)
    - You can define new object types (using classes), but you can't define new primitive types

Declarations and Types

- You can define new object types (using classes), but you can't define new primitive types

```
Ex :     int num = 123;                               //this is a primitive type

     String newString = new String("123");   //this is an object type
```

- Notice the case of names in two types. Primitive type starts with a lowercase letter. By convention, object type names should always have all the internal names starting with uppercase letters.

Static Typing

- Static vs. dynamic
  - Static or compile-time means “known or done before the program runs”
  - Dynamic or run-time means “known or done while the program runs”
- Java has static typing
  - Expressions are checked for type errors before the program runs

```
int n = 1;
n = n + “2”; // type error ( compile time error )
```

- Python has dynamic typing
  - it wouldn’t complain about n + “2” until it reaches that line in the running program

Strings

- A String is an object representing a sequence of characters
- Strings can be concatenated using +
  - “8” + “4” -> “84”
- String objects are immutable (never change), so concatenation creates a new string, it doesn’t change the original string objects
- String objects have various methods
  - String seq = “4,2,1”;
  - seq.length() -> 3
  - seq.charAt(0) -> ‘4’
  - seq.substr(0, 2) -> “4,”
- Use Google to find the Java documentation for String – Learn how to read the Java docs, and get familiar with them

Strings - Example

```
public static void main( String[] args ) {
        string str1 = “apple”;
        string str2 = “An”;
        System.out.println( str2 + “ ” + str1 );          // output →        An apple
}
```

Arrays

- array is a fixed-length sequence of values
- Base type of an array can be any type (primitive, object, another array type)
  - int[] intArray; char[] charArray; String[] stringArray;
  - double[][] matrix;   // array of arrays of floating-point numbers
- Fresh arrays are created with new keyword
  - int Array a = new int[5]; // makes array of 5 integers
- Operations on an array
  - int Array[0] = 200;   //sets a value
  - Int Array[0]   // gets a value -> 200
  - Int Array.length -> 5 // gets array's length
- Unlike a String, an array's elements can be changed
- But once created, an array's length cannot be changed
- So it's not like a Python list – a Java array can't grow or shrink

Arrays - Example

```
public static void main( string[] args ) {
        int array[5] = { 1,2,3,4,5 };
        int length = array.length;
        array[length-1] = 8;

        System.out.println( array[4] + array[2] );      //output →       11 ( 8 + 3 )
}
```

Arrays – Example (Max value of an array)

- Returns the maximum value of an array of positive integers. Returns 0 if the array is empty.

```
public static int maxValue(int[] array) {
        int max = 0;
        for (int i = 0; i < array.length; ++i) {
                if (array[i] > max)     max = array[i];
        }
        return max;
}
```

Variables

- Variables are of three types
  - Static variables
  - Instance variables
  - Local variables
- **Static Variable** and its value is common to all instances of a class. It is a class level variable

```
public class Zebra {

        private static string skintexture = “Black and White stripes”;

        private string getSkinTexture() {
                return Zebra.skintexture;        //this will be the same for all Zebras ( Zebra
instances )
        }
}
```

Variables

- **Local Variable** and its value is common only to the block, which it is being declared.
- Also known as automatic variables.
- These are created while a block or method is active and destroyed when the block or method exits.

```
public class Dog {

        private void setSkinColour(String clr) {
                String skinColour = clr + “ colour”;          //change depend on the parameter passed
                                                      //will set doggyColour = brown colour if the value
                                                      //passed is “brown”

        }
}
```

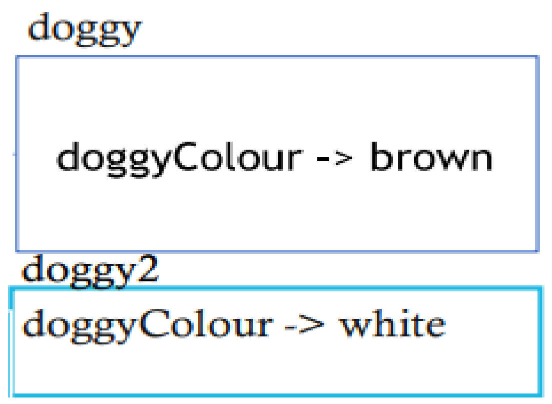
Variables

- **Instance Variable** and its value is common only to the object instance, which it is being declared.

```
public class Dog {

      private String doggyColour = “brown”;

      public static void main ( String[] args ) {
            Dog doggy = new Dog();
            System.out.println( doggy.doggyColour );      //output ✎      Brown;
            Dog doggy2 = new Dog();
            doggy2.setDogColour("White");
            System.out.println(doggy2.doggyColour);
      }
      private setDogColour(String colour){
            doggyColour = colour;
            }
}
```



Both are instances of Dog class

Methods

- Operations on the data within a class is carried out using methods
  - For example the Math class methods enable the user to do math operations such as finding square root (Math.sqrt(4))
- Variables within methods are called local variables (encapsulated)
- A method definition has the following syntax

  -
    ```
    return-value-type method-name(parameter-list) {
            declarations and statements;
     }
    ```
- Method definition should appear inside a class definition
- The return statement should have a compatible type

Methods

- Methods are of three types
  - Static methods
  - Instance methods
- **Static Methods** are executed at the compilation time. All the instances of that object type has the same effect of the static method.

```
 public class Dogc{

        private static string doggyColour = “”;

        private static void setSkinColour() {
                doggyColour = “brown”;                    //will set doggyColour = brown colour at the
beginning of                                                    //compilation
        }
 }
```

Methods

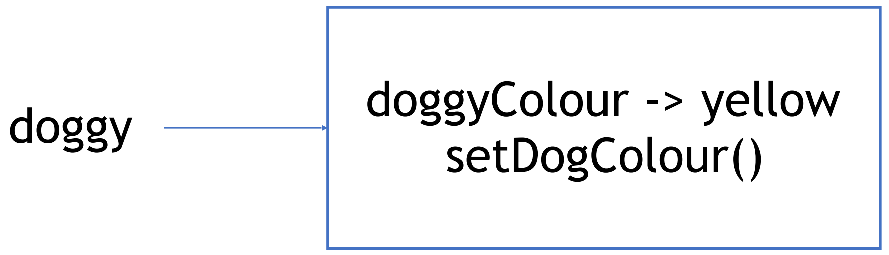
- **Instance Methods** can be called via any of the instances, with any parameter value dynamically.

```
public class Dog {

        private string doggyColour;

        public setDogColour( string colour ) {
                doggyColour = colour;
        }

        public static void main ( string[] args ) {
                Dog doggy = new Dog();
                doggy.setDogColour( “Yellow” );
                System.out.println( doggy.doggyColour );     //output →      Yellow;
        }
}
```



Constructor

- Special type of method that initialize an object
- Constructor name should be same as class name and return type should be void
- Default constructor requires no arguments, however you can define constructors with one or more arguments
- Can define access modifiers, default – same access level as class

```
public class Dog {

    public Dog() {
        // initialize
    }
}
```

Example – Instance object

```
public class Dog {

    private String dogColour;
    private int age;

    public Dog(dogColour, age) {
        this.dogColour = dogColour;
        this.age = age;
    }

    public static void main (String []args) {
        Dog doggy = new Dog();
        Dog doggy2 = new Dog(“brown”, 5);
    }
}
```

Control Structures

- Control structures are used to deviate from the sequential operation of programs.
- **If/else** structure and **if/else if** structures
- **while** repetition structure, **do/while** structure (Different from **while**)
  - When x=2;
    post increment x++ //prints 2;
    pre increment ++x //displays 4 useful in control structures
- **for** control structure
- **switch** structure. Selection among multiple directions
- **break** and **continue** control statements. Break exits from a loop or switch statement. continue moves to the end and loops back (skips remaining)

Control Structures

```
if(condition) {
      // expression if condition is true
} else {
      // expression if condition is false
}
```

```
if(condition1) {
      // expression if condition1 is true
} else if(condition2) {
      // expression if condition is true
} else {
      // expression if conditions are false
}
```

```
for(int i; i < value; i++ ) {
      // print i
}
```

```
for( data_type name: array) {
      // statements
}
```

Control Structures

```
while(boolean_expression) {
	// statements
}
```

```
do {
	// statements
} while(boolean_expression) ;
```

```
switch(expression) {
	case value1 :
		// statements
		break //optional
	case value2 :
		// statements
		break //optional
	default :	// optional
		// statements
}
```

Duration and Scope of Identifiers

- Identifiers are used for variable and reference names.
- The duration of an identifier is the period during which that identifier exists in memory.
- An identifiers scope is where the identifier can be referenced in a program.
- Instance variables of a class are automatically initialized by the compiler if the programmer does not provide initial values. All primitive data types are initialized to zero and boolean variables to false.
- However Automatic variables need to be initialized before they can be used

Duration and Scope of Identifiers

- Variables of static duration remain in memory after creation till the program terminate
- Their storage is allocated and initialized when their classes are loaded into memory
- Duration of a static variable has nothing to do with the scope. Even though it exists in memory it may not be accessible in some cases
- Methods and instance variables have class scope. That is methods and instance variables are global within the class.
- Identifiers within a block have block scope. That is within the braces of a block. The variable scope remains the same for nested blocks too
- If a local variable in a method has the same name as an instance variable, the instance variable is hidden until the block terminates execution

Duration and Scope of Identifiers

- For labels used with break and continue, we have a special scope called method scope. That is the label is visible to only the method in which it is used
- A variable name in an inner block having the same name as a variable in an outer block will cause a syntax error.
- As a general practice you should avoid defining local variables that hide instance variable

Variable Naming Convention

- Variable names are case-sensitive.
- An unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "$", or the underscore character "_".
- The convention is to always begin your variable names with a letter, not "$" or "_".
- Auto-generated names will but your variable names should always avoid using it.
- It's technically legal to begin your variable's name with "_", this practice is discouraged.
- White space is not permitted.
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- Use full words instead of cryptic abbreviation
- The name must not be a keyword or reserved word.

Variable Naming Convention

- If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word (Camel Case)
  - Eg: gearRatio, currentGear
- If your variable stores a constant value, capitalizing every letter and separating subsequent words with the underscore character. By convention, the underscore character is never used elsewhere.
  - Eg: static final int NUM_GEARS = 6

Java Language Keywords

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

Object Oriented Programming

OOP

- Objects can be used effectively to represent real world entities
  - Warriors, monsters, trees, binoculars…..
- As another example, an object might represent a particular employee in a company
- Each employee object handles the processing and data management related to that employee
- Think of some other examples

OOP

- An object has:
  - state - descriptive characteristics
  - behaviours - what it can do (or what can be done to it)
- The state of a bank account includes its account number and its current balance
- The behaviours associated with a bank account include the ability to make deposits and withdrawals
- Note that the behaviour of an object might change its stat

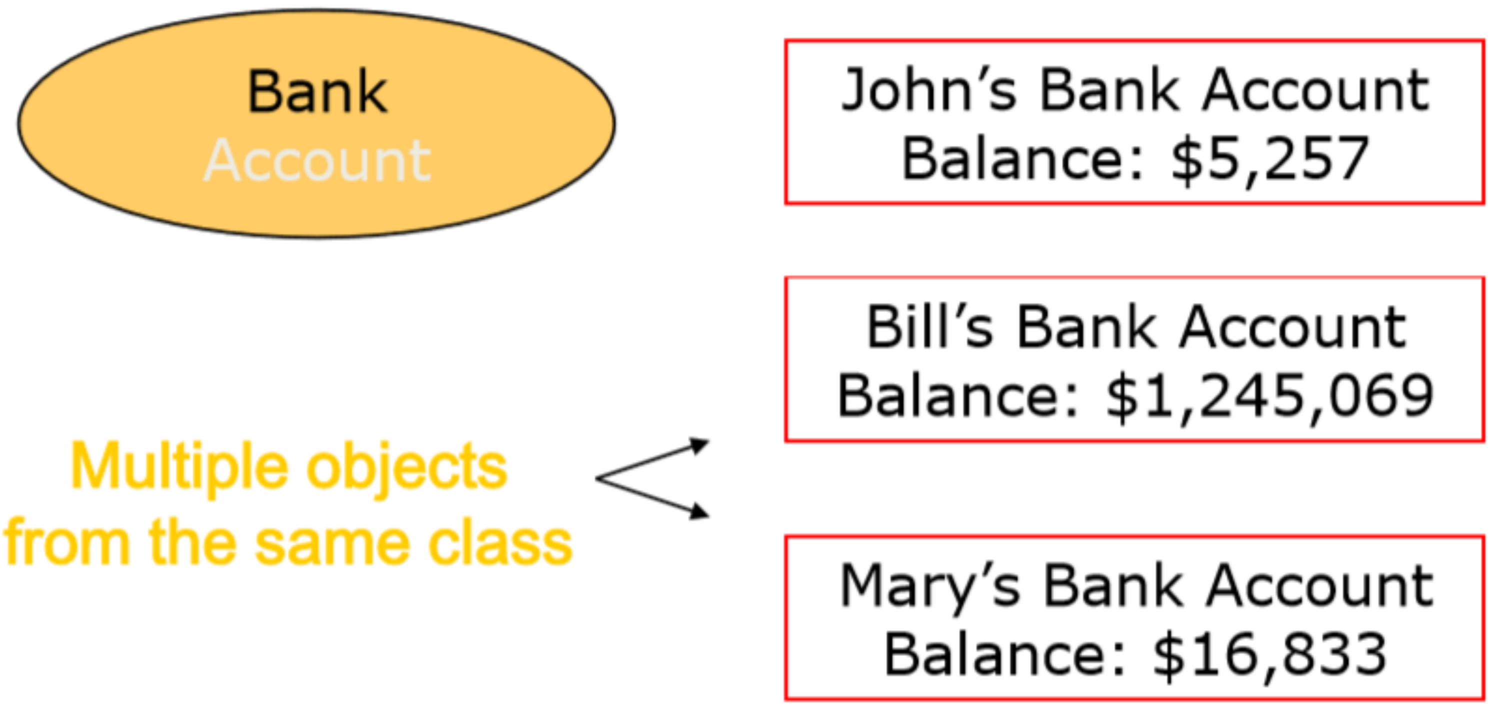
Classes

- An object is defined by a class
- A class is the blueprint of an object
- Multiple objects can be created from the same class

Objects vs Classes

A class
(the concept)

An object
(the realization)



Multiple objects
from the same class

Attributes

- Contain current state of an object
- Attributes can be classified as simple or complex.
- Simple attribute can be a primitive type such as integer or string, which takes on literal values.
- Complex attribute can contain collections and/or references.

Exercise

- Think of an object
    - What is the class it belongs to?
    - What are its attributes?
    - What are its methods?