OOP Concepts

Object-oriented programming

- Revolves around the concept of **objects** as the basic elements of your programs.
- These objects are characterized by their **properties** and **behaviors**.

Object	*Properties*	*Behavior*
Car	type of transmission manufacturer color	turning braking accelerating
Lion	Weight Color hungry or not hungry tamed or wild	roaring sleeping hunting

Encapsulation

- concealment of an object's data and methods from outside sources
  - information hiding - data
  - implementation hiding – methods
- hide specific object attributes and methods from outside sources and provides the security that keeps data and methods safe from inadvertent changes
- enclosure of data and methods within an object
- to treat all of an object's methods and data as a single entity

Instantiation

- Creation of objects from classes (written by programmers, java predefined classes)

Polymorphism

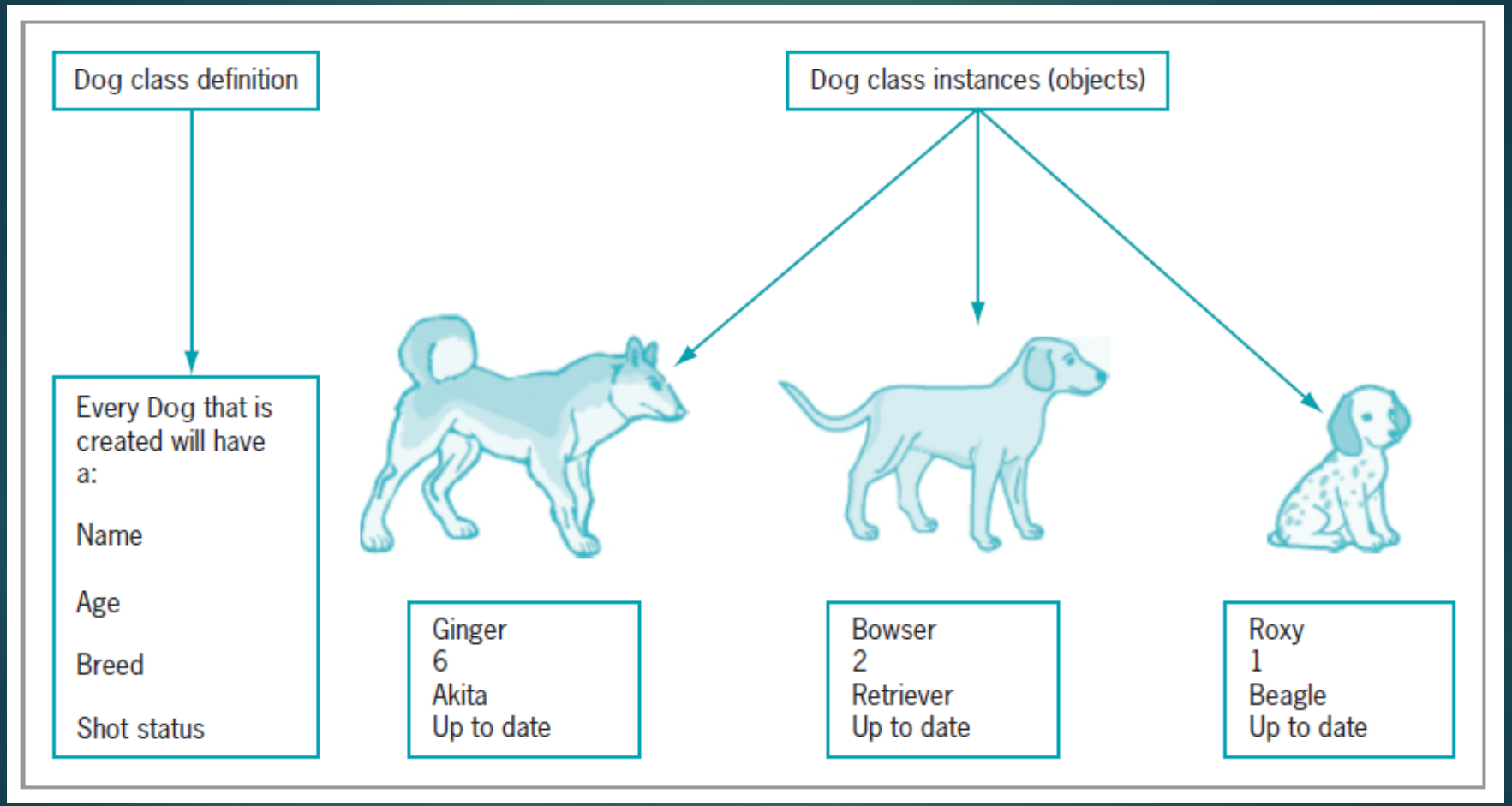
- means “many forms”
- it describes the feature of languages that allows the same word or symbol to be interpreted correctly in different situations based on the context

Inheritance

- the ability to create classes that share the attributes and methods of existing classes but with more specific features

Class and Object

- Class
  - a template, a prototype or a blueprint of an object
  - the fundamental structure in object-oriented programming
- Class contains
  - Fields (properties or attributes)
    - Specifiy the data types defined by the class
  - Methods (behaviour)
    - Specify the operations
- Object
  - instance of a class
  - Composed of a set of data (field) describing the characteristic of the object and set of methods (behavior) that describes how the object behaves


Dog class definition
Dog class instances (objects)
Every Dog that is created will have a:
Name
Age
Breed
Shot status
Ginger
6
Akita
Up to date
Bowser
2
Retriever
Up to date
Roxy
1
Beagle
Up to date

Car Class	Object Car A	Object Car B
Plate Number	ABC 111	XYZ 123
Color	Blue	Red
Manufacturer	Mitsubishi	Toyota
Current Speed	50 km/h	100 km/h
Accelerate Method		
Turn Method		
Brake Method		

- Classes provide the benefit of **reusability**.
- Software programmers can use a class over and over again to create many objects.

Creating a Class

```
<modifier> class <name> {
        <attributeDeclaration>*
        <constructorDeclaration>*
        <methodDeclaration>*
}
```

- An optional access modifier
- The keyword class
- Any legal identifier you choose for the name of your class

```
public class StudentRecord {
    //additional codes
}
```

- where, public - means that our class is accessible to other classes outside the package
- class - this is the keyword used to create a class in Java
- StudentRecord - a unique identifier that describes our class

Coding Guidelines

- Think of an appropriate name for your class. Don't just call your class XYZ or any random names you can think of.
- Class names should start with a CAPITAL letter.
- The filename of your class should have the SAME NAME as your public class name.

Declaring Attributes

- <modifier> <type> <name> [= <default_value>];
- public class StudentRecord {

```
    private String name;
    public int age;
    protected double grade;
    //additional codes
}
```

Coding Guidelines

- Declare all your instance variables on the top of the class declaration.
- Declare one variable for each line.
- Instance variables, like any other variables should start with a SMALL letter.
- Use an appropriate data type for each variable you declare.
- Declare instance variables as private so that only class methods can access them directly.

Instance (nonstatic) variables

```
public class StudentRecord {
    private String name;
    //additional code
 }
```

- Each object gets its own copy of each nonstatic data field.

Class (static) variables

- ```
  public class StudentRecord {
      //instance variables declared
      private static int studentCount;
      //additional code
  }
  ```
- variables that belong to the whole class
- only one value would be shared by all objects
  ```

Car Class		Object Car A	Object Car B
Instance Variables	Plate Number	ABC 111	XYZ 123
	Color	Blue	Red
	Manufacturer	Mitsubishi	Toyota
	Current Speed	50 km/h	100 km/h
Class Variable	Count = 2		
Instance Methods	Accelerate Method		
	Turn Method		
	Brake Method		

Methods

```
<modifier> <returnType> <name>(<parameter>*) {
        <statement>*
    }
```

- <modifier> - can carry a number of different modifiers
- <returnType> - can be any data type (including void)
- <name> - can be any valid identifier
- <parameter> - <parameter_type> <parameter_name>[,]

Static Methods

- methods that can be invoked without instantiating a class (means without invoking the new keyword)
- belong to the class as a whole and not to a certain instance (or object) of a class
- distinguished from instance methods in a class definition by the keyword static
- ```
  public static void calculate(int x){
  }
  ```
- ```
  Classname.staticmethod(parameters)
  ```

Instance (non-static) Methods

- methods used with object instantiations
- objects can use either static or non-static but only nonstatic methods behaves uniquely for each object
- You cannot use a nonstatic method without an object

Static	Non-static
static is a keyword	**No** keyword used
class variables / static variables	instance variable / non-static variable
Class methods / static methods	Instance methods/ non-static methods
No need for objects	**Must** use an object
instantiate 10 objects, only **one copy** of that field exists in memory	instantiate 10 objects, **10 copies** of that field exists in memory
instantiate 10 objects, only one copy of the method exist and method **does not** receive a **this** reference	Instantiate 10 objects, only one copy of the method exist but the method receives a **this reference** that contains the address of the object currently using it
every instance of a class shares the same copy of any class variables	separate memory location for each nonstatic field in each instance

Instantiation

- To create an object or an instance of a class, we use the new operator.
- `<classname> <object> = new <constructor>`
- `Student one;`

  `one = new Student;`
- `Student one = new Student();`
- new operator
  - allocates a memory for that object and returns a reference of that memory location to you.
  - When you create an object, you actually invoke the class' constructor.
- Constructor
  - is a method where you place all the initializations
  - it has the same name as the class.

Mutator Methods

- methods that set or change field values
- conventionally start with the prefix ***set***

```
private int empNum;
public void setEmpNum(int emp){
     empNum = emp;}
```

Accessor Methods

- methods that retrieve values
- conventionally start with the prefix ***get***

```
private int empNum;
public int getEmpNum(){
    return empNum;
}
```

Constructor

- To declare a constructor

```
<modifier> <className> (<parameter>*) {
        <statement>* }
```

- Example

```
private int empNum;
public StudentRecord(){
   empNum = 12;
}
```

- A constructor establishes an object
- default constructor
  - does not require arguments
  - created automatically (implicit constructor)by the Java compiler for any class you create whenever you do not write your own constructor
- must have the same name as the class it constructs

- constructors cannot have a return type
- declared as public
- If you write a constructor (explicit constructor) for a class, you no longer have access to the automatically created version
- initial values to an object's data fields for the implicit default constructor:
  - Numeric - 0
  - Character - Unicode '\u0000'
  - Boolean - false
  - Fields that are object references (String etc.) - null

```
public class Sample{
   //attribute
   private String firstName;

   //constructor
   public Sample(){
     firstName = “Juan”;}

   //mutator
   public void setFirstName(String temp){
     firstName = temp;}

   //accessor
   public String getFirstName(){
     return firstName;}
}
```

```
public class Test{
  public static void main(String args[]){
    //instantiation
    Sample one = new Sample();
    Sample two = new Sample();

    //mutator and accessor applied
    one.setFirstName(“Hydi”);
    System.out.println(one.getFirstName);

    //constructor applied
    System.out.println(two.getFirstName);
  }
}
```