Objects and Classes

Core Concepts

- Java is an Object-Oriented Language and supports the fundamental concepts of OO (Polymorphism, Inheritance, Encapsulation, Objects, Classes, etc.)
- Object: Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.
- Class: A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

Defining a Class

- Objects with the same type belong to the same class
- A class includes a special method, a constructor
  - Used to initialise an object of the class as it is being created
- A class can contain any of the following variable types:
  - Local variables
  - Instance variables
  - Class variables
- A class can have any number of methods to acces the value of various kinds of methods.

Contructors

- Every class has a constructor, if we don't create one, the Java compiler builds a default constructor for that class.
- Pseudo-method to initialise newly created object.
- Each time a new object is created at least one constructor will be invoked.
- Main rule: same name as the class
- More than one constructor can be used per Class with Overloading.

```
public class Cat {
    private String name;
    private int age;

    public Cat() {
        name = "Tom";
        age = 77;
    }

    public Cat(String n) {
        name = n;
        age = 6;
    }

    public Cat(String n, int a) {
        name = n;
        age = a;
    }

    public void setName(String n) {
        name = n;
    }

    public void setAge(int a) {
        age = a;
    }

    public void showCatDetails() {
        System.out.println("The name of the cat is " + name);
        System.out.println("The age of the cat is " + age);
    }

}
```

Creating an Object

- Objects contain:
  - Identity
  - State
  - Behavior
- Creating an object: new operator
- There are three steps when creating an object from a class:
  - Declaration
  - Instantiation
  - Initialization

```
public class HouseHold {

    public static void main(String[] args) {

        Cat cat;
        cat = new Cat("Cirmi");
        Cat cat2;
        cat2 = new Cat("Ella", 8);
        Cat cat3;
        cat3= new Cat();

        cat.showCatDetails();
        cat2.showCatDetails();
        cat3.showCatDetails();
    }

}
```

Object Initialization

- Objects are created with the new operator
- Attributes are set to default values automatically:
  - 0 for numeric/char primitive types
  - false for boolean
  - null for objects
- Class-specific initial values then are set
- Constructor is called for more detailed initialisation

Parameter Passing

- Parameters refer to the list of variables in a method declaration:
  - Arguments are the actual values that are passed in when the method is invoked
  - When a method is invoked, the arguments used must match the declaration's parameters in type and order
- Parameter types:
  - Any data type for a parameter of a method or a constructor
  - Java programming language doesn't let you pass methods into methods
  - An object can be passwd into a method and then the object's methods be invoked
- A parameter is declared to a method or a constructor
  - A name is provided for the parameter
  - The name is used within the method body to refer to the passed-in argument
  - The name of a parameter must be unique in its scope

- Passing primitive data type arguments:
  - Passed into methods by value
  - Copy of the primitive is sent into the method
- Passing reference data type arguments
  - Passed into methods by value
  - Copy of the reference is sent into the method
  - Changing the reference value in the method's body won't affect the value on the caller's side
  - Changes in the values of the object's fields are visible on the caller's side

The Current Object

- With the this keyword
- Available in constructors and methods, but not in static methods
- Often used to avoid possible ambiguity

```
public Cat(String name, int age) {
    this.name = name;
    this.age = age;
}
```

Class Members

- Use of the static keyword
  - Create fields that belong to the class, rather than to an instance of the class
  - Create methods that belong to the class, rather than to an instance of the class
- Class Variable
  - Common to all objects
  - Class variables are referred to by the class name itself, as in: Dog.numberOfDogs
- Class Methods
  - They have the static modifier in their declarations
  - Should be invoked with the class name
  - Without the need for creating an instance of the class
  - Class methods can access class variables and class methods directly
  - Class methods cannot access instance variables or instance methods directly

```java
public class Dog {

    private String name;
    private String breed;
    private int id;
    public static int numberOfDogs = 0;

    public Dog(String name, String breed) {
        this.name = name;
        this.breed = breed;
        id = ++numberOfDogs;
    }

    public void showDogDetails() {
        System.out.println("The name of the dog is: " + name + ", the breed of the dog is: " + breed);
    }

    public static int getNumberOfDogs() {
        return numberOfDogs;
    }
}
public class HouseHold {

    public static void main(String[] args) {

        Dog dog;
        dog = new Dog("Cujo", "St. Bernard");
        Dog dog2 = new Dog("Kántor", "German Sheperd");
        dog.showDogDetails();
        dog2.showDogDetails();
        System.out.println("The number of dogs with class method: " + Dog.getNumberOfDogs());
        System.out.println("The number of dogs with class variable: " + Dog.numberOfDogs);
    }

}
```