# Objects and Reference Variables

Maria-Iuliana Dascalu
mariaiuliana.dascalu@gmail.com

FILS, UPB, 2011
Programming Languages

# Related Notions

- Object: instance of a class
- Class: blueprint from which objects are made
- Instantiation: process of creating objects from a class
- Reference value: returned when an object is created
- Reference variable(object reference): a variable that can store a reference value

# Steps in Objects' Creation

Example: *Dog myDog= new Dog ();*

- Declare a reference variable of a class:

*Dog myDog = new Dog();*

- Create an object:

*Dog myDog = new Dog();*

- Assign the object to the reference:

*Dog myDog = new Dog();*

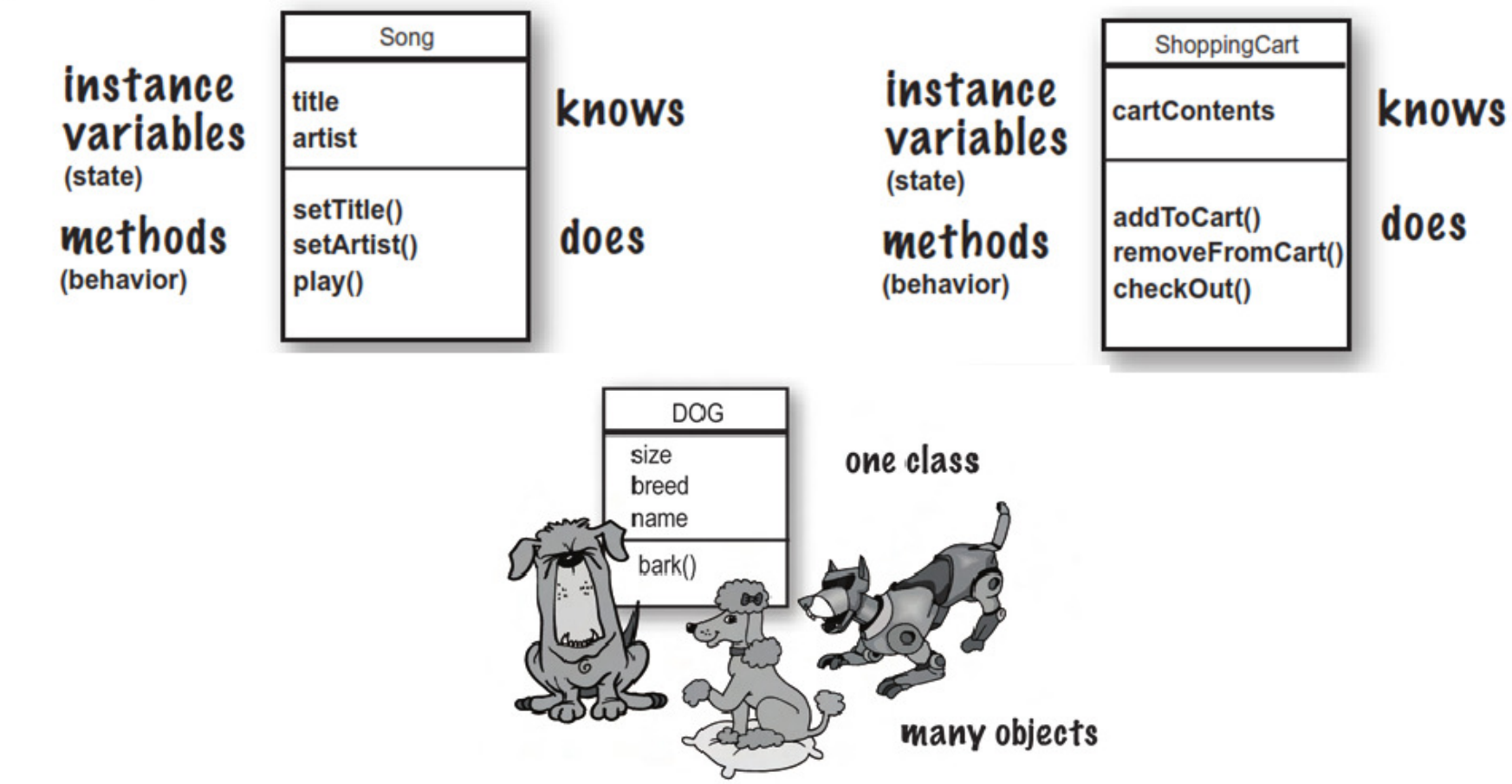
# Characteristics of objects

- Behavior (things what object does): what methods you can apply to it?/ what can you do with the object?

- State (things what object know): : how does the object react when you apply these methods?

- Identity: how is the object distinguished from others having the same behavior and state?

# Examples




- Implement the java class Dog: write down its instance variables and its methods!

# The Dog Class

```
7   class Dog
8   {
9       // instance variables
10      private int size;
11      private String breed;
12      private String name;
13
14      // methods
15      public void bark()
16      {
17          System.out.println("The dog "+this.name+" barked");
18      }
19  }
20
21  public class TestDog
22  {
23      public static void main(String[] args)
24      {
25          // instantiation
26          Dog myDog = new Dog();
27          myDog.bark();
28      }
29  }
```

What would display the program? (Pay attention to the default constructor!)

# The Constructor

- It runs before the object can be assigned to a reference
- Has the same name as the class
- Resembles a method, but has no return type
- It is used to initialize the state of an object

```
public class Dog()
{
    public Dog ()
        {
            size = 13;
            Breed = “cocker”;
            name = “Micky”;
        }
}
```

# Overloaded Constructors

- If you have more than one constructors in a class, they must have different argument lists!
- Pay attention: both the variable type and variable order matters!

- If you write a constructor that takes arguments, and you still want a no-arg constructor, you'll have to build the no-arg constructor by yourself!

# Overloaded Constructors

```java
// constructors
public Dog()
{
    this.size= 13;
    this.breed = "cocker";
    this.name = "Micky";
}

public Dog(int size, String breed, String name)
{
    this.size= size;
    this.breed = breed;
    this.name = name;
}
```

```java
public class TestDog
{
    public static void main(String[] args)
    {
        // instantiation
        Dog myDog = new Dog();
        myDog.bark();
        Dog yellowDog = new Dog(15, "Golden Retriever", "Goldy");
        yellowDog.bark();
    }
}
```

# Accesibility

```
class Dog
{
    // instance variables
    private int size;
    String breed;
    private String name;
    // methods
    public void bark() {System.out.println("The dog "+this.name+" barked");}
    // constructors
    public Dog(int size, String breed, String name)
    {
        this.size= size;
        this.breed = breed;
        this.name = name;
    }
}
public class TestDog
{
    public static void main(String[] args)
    {
        Dog yellowDog = new Dog(15, "Golden Retriever", "Goldy");
        System.out.println("The dog "+yellowDog.name+" is a "+yellowDog.breed);
    }
}
```

public

RuntimeException: Uncompilable source code - name has private access in testdog.Dog

Has access, as "breed"is public

To solve the problem, implement a getter (getName method) for the class Dog!

# Static members *vs.* instance members

- Static members belong to a class, not to any individual objects.
- Static members can be accessed both by the class name and via object references.
- Instance members can only be accessed by object references.

# Example

```
class Dog
{
    String name;
    int id;
    static int nextId = 1;
    public Dog(String name)
    {
        this.name = name;
        this.id = nextId;
        nextId++;
    }
}
public class TestDog
{
    public static void main(String[] args)
    {
        Dog yellowDog = new Dog("Goldy"); Dog blackDog = new Dog("Blacky");
        System.out.println("The dog "+yellowDog.id+" has the name "+yellowDog.name);
        System.out.println("The dog "+blackDog.id+" has the name "+blackDog.name);
        int dogsNo = Dog.nextId-1;
        System.out.println("The total numbers of dogs is: "+dogsNo);
    }
}
```

yellowDog.nextId or blackDog.nextId returns the same result!

# Exercises

What do the following sequences of code print? Explain!

```
class Example
{
    static int x=0;
    Example() {x++;}
}

public class JavaApplication30 {
    public static void main(String[] args) {
        Example a = new Example();
        Example b = new Example();
        System.out.println(a.x);
        System.out.println(b.x);
        a.x = 100;
        b.x = 200;
        System.out.println(a.x);
        System.out.println(b.x);
    }
}
```

```
class C1
{
    static int x=1;
    int y=2;
    static C1() {x=4;}
    C1 (int y){
        this.y = y+1;
    }
}

public class JavaApplication30 {
    public static void main(String[] args) {
        System.out.println(C1.x);
    }
}
```

# Objects' Identity

- Two objects are identical if they have the same spot in memory ("reference equality“)”: a change to one will affect the other.
- It is implemented with “==“ operator.
- Test the following piece of code and discuss it!

```
public class StringIdentity {
  public static void main(String[] args) {
    String a = new String("Hello");
    String b = new String("Hello");

    System.out.println(a == b);
    System.out.println(a.equals(b));
  }
}
```

# Memory in Java

- 2 areas of the memory
  - Stack (where method invocations and local variable live)
  - Heap (where objects live)

  Note:

  Instance variable are declared inside a class, not inside a method; they live inside the object they belong to.

  The Java runtime environment deletes objects when it determines that they are no longer being used (garbage collection).

# Exercises

Ex1: Implement a java class, named Product. Each product has a name and a price. Write constructors for the class Product and the following methods: getPrice() and setPrice(). Then write a Shop class to test your work.

Ex 2: Create a BankAccount class, with two private instance variables, named accountNumber and accountSum. Write constructors for the BankAccount class and a method for extracting money from a bank account. Then write a Bank class to test your work.