

Boaz Kantor

Introduction to Computer Science,

Fall semester 2010-2011

IDC Herzliya

# Classes, objects, references, encapsulation and whatnot

# Agenda

- Object state (4 to 6)
- Methods and parameters (7 to 8)
- Visibility and encapsulation (9 to 10)
- Return values (11 to 13)
- Aliases (14 to 15)
- Constructors (16 to 19)
- Static (20 to 28)
- Standard methods (29 to 31)
- String vs. StringBuilder (32)

# Our mission

- Write a class called “Dog”
- The class will represent a real-life dog
- The dog has a breed, age and name
- Implement constructors, getters & setters

# Where to begin?

1. Write a skeleton:

```
public class Dog {  
    String breed;  
    String name;  
    int age;  
}
```

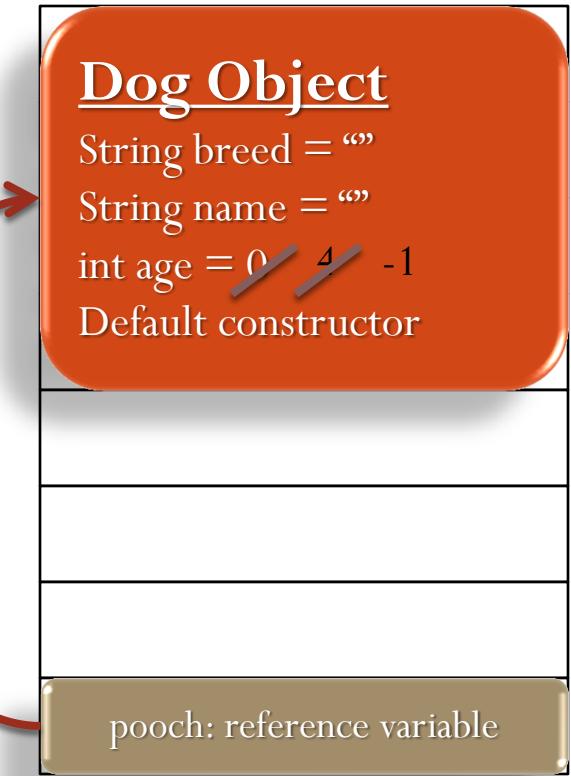
2. Write a driver:

```
public class Owner {  
    public static void main(String[] args) {  
        Dog pooch = new Dog();  
    }  
}
```

# How it looks in memory

```
public class Dog {  
    String breed;  
    String name;  
    int age;  
}
```

```
// in the main() method of class Owner:  
Dog pooch = new Dog();  
pooch.age = 4;  
pooch.age = -1;
```



# A method, parameters, what is this?

```
public class Dog {  
    String breed;  
    String name;  
    int age;  
    setAge(int age) {  
        if (age > 0 && age < 99) {  
            this.age = age;  
        }  
    }  
}
```

```
public class Dog {  
    String breed;  
    String name;  
    int age;  
    setAge(int age) {  
        if (age > 0 && age < 99) {  
            this.age = age;  
        }  
    }  
}
```

```
// in the main() method of class Owner:  
Dog pooch = new Dog();  
pooch.setAge(4);  
pooch.setAge(-1);  
pooch.age = -1;
```

## Dog Object

String breed = ""  
String name = ""  
int age = ~~0 / 4~~ -1  
Default constructor  
setAge(int age)

pooch: reference variable

# Visibility & encapsulation

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    public setAge(int age) {  
        if (age > 0 && age < 99) {  
            this.age = age;  
        }  
    }  
}
```

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    public setAge(int age) {  
        if (age > 0 && age < 99) {  
            this.age = age;  
        }  
    }  
}
```

// in the main() method of class Owner:

```
Dog pooch = new Dog();  
pooch.setAge(4);  
pooch.setAge(-1);  
pooch.age = -1;          // COMPIILATION ERROR!  
int poochsAge = pooch.age; // COMPIILATION ERROR!
```

## Dog Object

String breed = ""

String name = ""

int age = 0 ~~/~~ 4

Default constructor

setAge(int age)

# Return values

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    public void setAge(int age) {  
        if (age > 0 && age < 99) {  
            this.age = age;  
        }  
    }  
    public int getAge() {  
        return this.age;  
    }  
}
```

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    public void setAge(int age) {  
        if (age > 0 && age < 99) {  
            this.age = age;  
        }  
    }  
    public int getAge() {  
        return this.age;  
    }  
}
```

// in the main() method of class Owner:

```
Dog pooch = new Dog();  
pooch.setAge(4);  
int poochsAge = pooch.getAge();
```

## Dog Object

String breed = ""

String name = ""

int age = 4

Default constructor

setAge(int age)

int getAge()

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    public void setAge(int age) {  
    public int getAge() {  
    public void setName(String name) {  
    public String getName() {  
    public void setBreed(String breed) {  
    public String getBreed() :  
}  
}
```

// in the main() method of class Owner:

```
Dog pooch = new Dog();  
pooch.setAge(4);  
Dog joola = new Dog();  
joola.setAge(1);
```

### Dog Object

String breed = ""  
String name = ""  
int age = 4  
Default constructor  
Getters / setters

### Dog Object

String breed = ""  
String name = ""  
int age = 1  
Default constructor  
Getters / setters

joola: reference variable

pooch: reference variable

# Aliases

- When a reference variable X points to an object, and a reference variable Y points to the same object, we say that Y is an alias of X, that X is an alias of Y and that they are both aliases of each other.
- It's not always that you have to create aliases, and not always avoid them, but you should always be aware when aliasing occurs.
- This is because aliasing may compromise the encapsulation principles.

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    public void setAge(int age) {  
    public int getAge() {  
    public void setName(String name) {  
    public String getName() {  
    public void setBreed(String breed) {  
    public String getBreed() {  
}  
}  
  
// in the main() method of class Owner:  
Dog pooch = new Dog();  
pooch.setAge(4);  
Dog joola = pooch;  
joola.setAge(1);  
System.out.println(pooch.getAge()); // "1"  
System.out.println(joola.getAge()); // "1"
```

## Dog Object

String breed = ""

String name = ""

int age = ~~4~~ 1

Default constructor

Getters / setters

joola: reference variable

pooch: reference variable

# Constructors

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    public Dog(int age, String breed, String name) {  
        this.setAge(age);  
        this.setBreed(breed);  
        this.setName(name);  
    }  
    // getters and setters below  
    // setAge(int age), getAge()  
    // setName(String name), getName()  
    // setBreed(String breed), getBreed()  
}
```

```

public class Dog {
    private String breed;
    private String name;
    private int age;
    public Dog(int age,
              String breed
              String name) {
        this.setAge(age);
        this.setBreed(breed);
        this.setName(name);
    }
    // getters and setters
}

```

// in the main() method of class Owner:

```

Dog pooch = new Dog();           // COMPILED OK
Dog pooch = new Dog(4, "Boxer", "Pooch");
Dog joola = new Dog(1, "Mixed", "Joola");
System.out.println(pooch.getAge()); // "4"
System.out.println(joola.getAge()); // "1"

```

### Dog Object

String breed = "Boxer"  
 String name = "Pooch"  
 int age = 4  
 Dog(int, String, String)  
 Getters / setters

### Dog Object

String breed = "Mixed"  
 String name = "Joola"  
 int age = 1  
 Dog(int, String, String)  
 Getters / setters

joola: reference variable

pooch: reference variable

# More on constructors

```
public class Dog {  
    private String breed;  
    private String name;  
    private int age;  
    private static final int DEFAULT_AGE = 0;  
    private static final String DEFAULT_BREED = "unknown";  
    private static final String DEFAULT_NAME = "n/a";  
  
    public Dog() {  
        this.setAge(DEFAULT_AGE);  
        this.setBreed(DEFAULT_BREED);  
        this.setName(DEFAULT_NAME);  
        OR:  
        this(DEFAULT_AGE, DEFAULT_BREED, DEFAULT_NAME);  
    }  
  
    public Dog(int age, String breed, String name) {  
        this.setAge(age);  
        this.setBreed(breed);  
        this.setName(name);  
    }  
    // getters and setters below  
}
```

```

public class Dog {
    private String breed;
    private String name;
    private int age;
    public Dog() {
        this(DEFAULT_AGE, ...);
    }
    public Dog(int age,
              String breed
              String name) {
        this.setAge(age);
        this.setBreed(breed);
        this.setName(name);
    }
    // getters and setters
}

```

// in the main() method of class Owner:

```
Dog pooch = new Dog(4, "Boxer", "Pooch");
```

```
Dog joola = new Dog();
```

© Boaz Kantor, IDC

## Dog Object

String breed = "Boxer"

String name = "Pooch"

int age = 4

Dog()

Dog(int, String, String)

Getters / setters

## Dog Object

String breed = "unknown"

String name = "n/a"

int age = 0

Dog()

Dog(int, String, String)

Getters / setters

joola: reference variable

pooch: reference variable

# New exercise

- Our mission:
  - Write a class called Ball
  - Each ball has colors
  - Implement the following API:
    - Constructors, getters, setters, yada yada yada...
    - getShape() to return the shape of the ball

# We already know all this

```
public class Ball {  
    private String color = "";  
    public Ball(String color) {  
        this.color = color;  
    }  
    public String getShape() {  
        return "Round";  
    }  
}
```

```
public class Ball {  
    private String color = "";  
    public Ball(String color) {  
        this.color = color;  
    }  
    public String getShape() {  
        return "Round";  
    }  
}
```

// in the main() method of class Owner:

```
Ball basketball = new Ball("Brown");  
Ball tennisBall = new Ball("Green");  
String basketBallShape = basketball.getShape();  
String tennisBallShape = tennisBall.getShape();  
System.out.println(basketBallShape); // "Round"  
System.out.println(tennisBallShape); // "Round"
```

### Ball Object

String color = "Brown"  
Ball(String color)  
String getShape()

### Ball Object

String color = "Green"  
Ball(String color)  
String getShape()

tennisBall

basketball

# Static methods

```
public class Ball {  
    private String color = "";  
    public Ball(String color) {  
        this.color = color;  
    }  
    public static String getShape() {  
        return "Round";  
    }  
}
```

```
public class Ball {  
    private String color = "";  
    public Ball(String color) {  
        this.color = color;  
    }  
    public static String getShape() {  
        return "Round";  
    }  
}
```

// in the main() method of class Owner:

```
Ball basketball = new Ball("Brown");  
Ball tennisBall = new Ball("Green");  
String ballShape = Ball.getShape();  
System.out.println(ballShape); // "Round"
```



# Static members

- All static members (variables, methods, constants) are not coupled with any object.
- Accessing (public) static members is by calling the class name:
  - `ClassName.staticVariable;`
  - `ClassName.staticMethod();`

# Example – what does it do?

```
public class RcSample {  
    private static int references = 0;  
    public RcSample() {  
        RcSample.references++;  
    }  
    public static int getReferences() {  
        return RcSample.references;  
    }  
}
```

# Another example

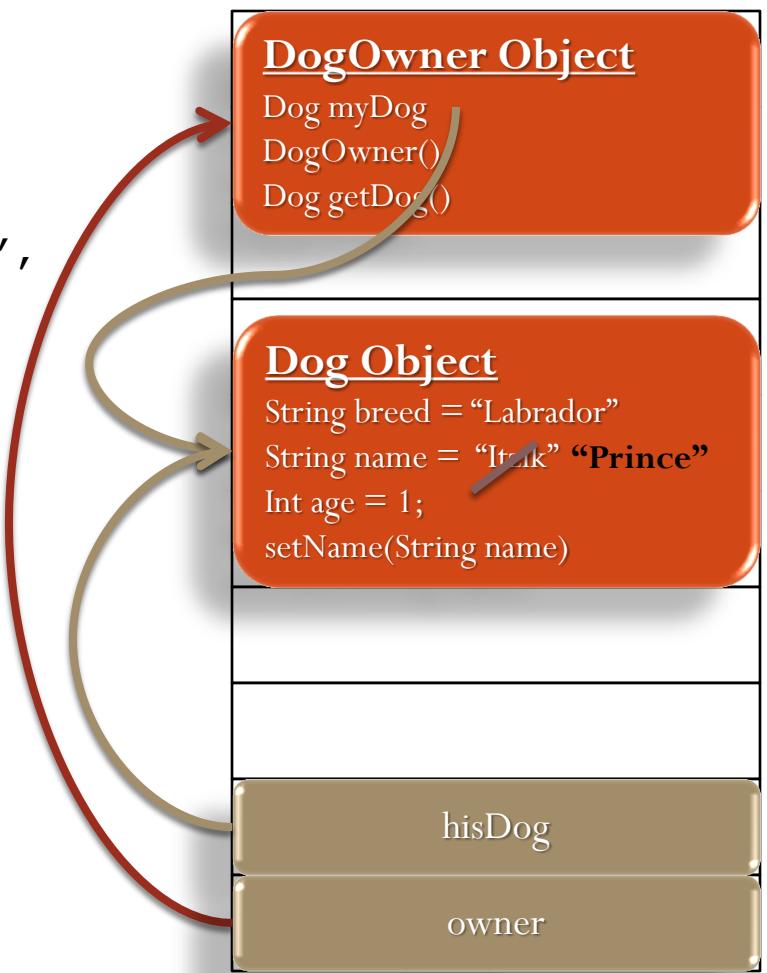
```
public class DogOwner {  
    private Dog myDog;  
    public DogOwner() {  
        myDog = new Dog(1, "Labrador", "Itzik");  
    }  
    public Dog getDog() {  
        return myDog;  
    }  
}  
  
// in another class..  
DogOwner d = new DogOwner();  
Dog hisDog = d.getDog();  
hisDog.setName("Prince");
```

```

public class DogOwner {
    private Dog myDog;
    public DogOwner() {
        myDog = new Dog(1,
                        "Labrador",
                        "Itzik");
    }
    public Dog getDog() {
        return myDog;
    }
}

// in another class..
DogOwner owner = new DogOwner();
Dog hisDog = owner.getDog();
hisDog.setName("Prince");

```



# clone()

- Aliasing out private members with external variables compromises encapsulation.
- Instead of aliasing we should implement a `clone()` method, which creates a copy of that member.

# Standard methods

```
public class Dog {  
    private int age;  
    private String breed;  
    private String name;  
    public Dog(int age, String breed, String name) {  
        setAge(age);  
        setBreed(breed);  
        setName(name);  
    }  
    public Dog clone() {  
        Dog newDog = new Dog(getAge(), getBreed(), getName());  
        return newDog;  
        OR:  
        return new Dog(getAge(), getBreed(), getName());  
    }  
}
```

# Standard methods

```
public Dog clone() {  
    return new Dog(getAge(), getBreed(), getName());  
}  
  
public String toString() {  
    return "Name: " + getName() + "\nBreed: " + getBreed() + "\nAge: " + getAge();  
}  
  
public boolean equals(Dog other) {  
    return getAge() == other.getAge()  
        && getBreed().equalsIgnoreCase(other.getBreed())  
        && getName().equalsIgnoreCase(other.getName());  
}  
  
public static Dog parse(String string) {  
    int age = Integer.parseInt(string.substring...); // find the age in the String  
    String breed = string.substring...; // find the breed in the String  
    String name = ... // find the name in the string  
    return new Dog(age, breed, name);  
}
```

# String vs. StringBuilder

- String
  - Is immutable = its state never changes:
    - `String bla = "abc";`
    - `bla = bla.replace("ab", "cc");`
  - Java does great effort to make it looks like you can change the text in a String variable.
  - Since it's immutable, you can't create an alias to it.
- StringBuilder:
  - Is mutable = its state is changed:
    - `StringBuilder bla = new StringBuilder("abc");`
    - `bla.replace("ab", "cc");`
  - To avoid Java's great effort we sometimes use StringBuilder instead of String.
  - You can create an alias to StringBuilder.
- Use StringBuilder whenever you need lots of string manipulation, but be extra careful with encapsulation and aliasing!