

Boaz Kantor
Introduction to Computer Science,
Fall semester 2009-2010
IDC Herzliya

Classes, objects, references, encapsulation and whatnot

1

© Boaz Kantor, IDC

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)

2

© Boaz Kantor, IDC

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

3

© Boaz Kantor, IDC

Where to begin?

1. Write a skeleton:

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

4 © Boaz Kantor, IDC

2. Write a driver:

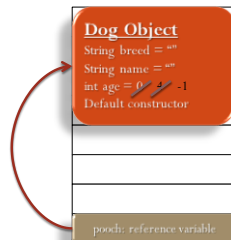
```
public class Owner {
    public static void main(String[] args) {
        Dog pooch = new Dog();
    }
}
```

5 © Boaz Kantor, IDC

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;
```



6 © Boaz Kantor, IDC

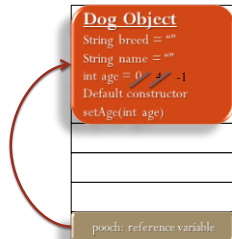
A method, parameters, what is this?

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

7 © Boaz Kantor, IDC

```
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;
```



8 © Boaz Kantor, IDC

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;
        }
    }
}
```

9 © Boaz Kantor, IDC

```

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; // COMPILATION ERROR!
int poochsAge = pooch.age; // COMPILATION ERROR!
    
```

Dog Object

String breed = ""

String name = ""

int age = 0 / 4

Default constructor

setAge(int age)

pooch: reference variable

10 © Boaz Kantor, IDC

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;
    }
}
    
```

11 © Boaz Kantor, IDC

```

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()

pooch: reference variable

12 © Boaz Kantor, IDC

```

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);
    
```

13 © Boaz Kantor, IDC

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.

14 © Boaz Kantor, IDC

```

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"
    
```

15

Constructors

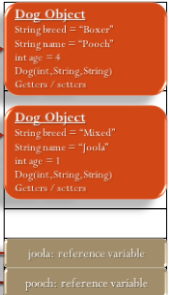
```
public class Dog {
    private String breed;
    private String name;
    private int age;
    public Dog(int age, String breed, String name) {
        this.setAge(age);
        this.setBreed(breed);
        this.setName(name);
    }
    // getters and setters below
    // setAge(int age), getAge()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

16 © Boaz Kantor, IDC

```
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();           // COMPILATION ERROR
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"
```

17 © Boaz Kantor, IDC



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
}
```

18 © Boaz Kantor, IDC

```

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();
    
```

19 © Boaz Kantor, IDC

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

20 © Boaz Kantor, IDC

We already know all this

```

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

21 © Boaz Kantor, IDC

```

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"
    
```

22 © Boaz Kantor, IDC

Static methods

```

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

23 © Boaz Kantor, IDC

```

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"
    
```

24 © Boaz Kantor, IDC

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();`

25

© Boaz Kantor, IDC

Example – what does it do?

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

26

© Boaz Kantor, IDC

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");
```

27

© Boaz Kantor, IDC

```

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");
    
```

28 © Boaz Kantor, IDC

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.

29 © Boaz Kantor, IDC

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 ());
    }
}
    
```

30 © Boaz Kantor, IDC

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(0, 4)); // Find the age in the String
    String breed = string.substring(5, 15); // Find the breed in the String
    String name = string.substring(16, 25); // find the name in the string
    return new Dog(age, breed, name);
}
```

31

© Boaz Kantor, IDC

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 look 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!

32

© Boaz Kantor, IDC

Exercise #5: The Students List

School: Efi Arazi School of Computer Science, IDC Herzliya
 Course: Introduction to Computer Science
 Author: Boaz Kantor

Exercise Targets

- Practicing class implementation when the API is given.
- Understanding the difference between API and implementation
- Understanding the importance of encapsulation

33

getting acquainted with the JDK's Builder
