
LETS LEARN JAVA
ABSTRACTION

Training Plan - Intermediate

Topics to be covered
What is abstraction, what are its usage Extending abstract class and overriding abstract methods Instantiation of classes 100% abstraction - Interfaces Implementing multiple inheritance using Interfaces

Abstraction

1. Used to define templates and use the template for driving implementation
2. Abstraction may be partial or complete (interfaces)
3. A class that has at least one abstract method needs to be defined as abstract
4. Implemented methods in abstract classes may still work as defaults
5. A class extending an abstract class is forced to implement all abstract methods unless the sub class is also defined as abstract. (Think of a template created on the basis of another template)
6. Objects cannot be instantiated for abstract classes

Example of an abstract class

```
public abstract class Pet implements Comparable<Pet> {

    protected String color;
    protected String name;
    protected Integer age;

    public abstract void run();

    public abstract void eat();

    public void gotVaccinated() {
        System.out.println("Pet got vaccinated today : " + LocalDate.now());
    }

    public Pet(String color, String name, Integer age) {
        this.color = color;
        this.name = name;
        this.age = age;
    }
```

The Pet class has now been made abstract. If you notice the class carefully, there are two methods `run()` and `eat()` which are declared, but which do not have any implementation (no logic in the methods).

Extending an abstract class

```
public class Dog extends Pet {

    private S
```

The type Dog must implement the inherited abstract method Pet.run()

2 quick fixes available:

- Add unimplemented methods
- Make type 'Dog' abstract

Press 'F2' for focus

```
    // Constr
    public Do                                    ng breed) {
        super
        this.breed = breed;
    }

    @Override
    public void eat() {
        if (breed.equalsIgnoreCase("German Shepherd")) {
            System.out.println("Dog is eating very fast");
        } else {
            System.out.println("Dog is eating and enjoying its food");
        }
    }
```

Since Dog is extending Pet, and Pet is marked abstract, hence Dog is forced to implement all the abstract methods of Pet class else the code will not compile.

Alternate – Define Dog as abstract and delegate the work of implementation of methods to the class that extends Dog.

```
    @Override
    public void run() {
        System.out.println("Dog is running really fast... as if its about to catch a bus");
    }
```

Creating instances of classes

```
public class PetList {
    public static void main(String[] args) {
        List<Pet> pets = new ArrayList<>();

        pets.add(new Pet("Munchkin", "White", 1));

        pets.add(new Dog("Bruno", "Brown", 3, "German Shepherd"));
        pets.add(new Dog("Tiny", "Black", 4, "Labrador"));
        pets.add(new Dog("Spooky Spider", "Striped", 2, "Golden Retriever"));
        pets.add(new Cat("Tim", "Gray", 3));
        pets.add(new Cat("Silky", "White", 2));
```

As you can see, Pet cannot be instantiated.

It actually makes sense. Pet has to be a something – a Dog, Cat, a Rabbit, a Mouse… But if you just say Pet, it does not mean much…

By using the abstraction concepts, you are telling Java that while all Pet have to have a name, a colour and an age, and some default behaviours, you will definitely need a specific implementation to work on it further

Complete abstraction - Interfaces

```java
public interface Playful {
    void jumpsInAir();

    void fetchesBall();

    void wagsTail();

}
```

```java
public interface Loveable {
    void makesCuteSounds();

    default void cuddles() {
        System.out.println("Cuddles and sleeps near you... You will feel so happy and warm");
    }

    void doesNotHarm();
}
```

Interfaces need not define access modifiers. By default they are considered public.

You may also have a default implementation of some methods in interfaces (Java 8 onwards), but you need to qualify them as **`default`**

Same way as you extend a class, you implement an interface. More than one interface may be implemented in one shot.

Implementing Interfaces

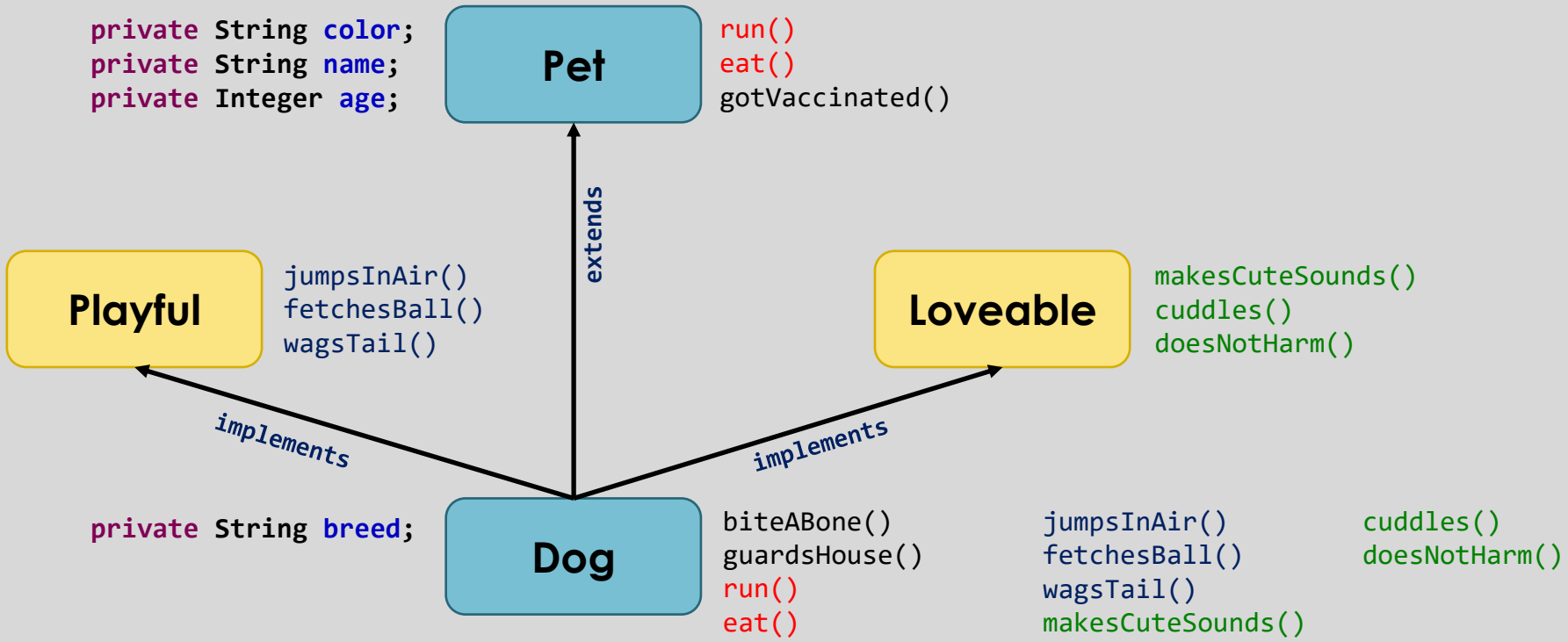
```java
public class Dog extends Pet implements Playful, Loveable {

    private String breed;

    @Override
    public void makesCuteSounds() {
        System.out.println("Who does not love the barking sound of a cute puppy");
    }

    @Override
    public void doesNotHarm() {
        System.out.println("Dog 'usually' to do not harm you, especially its owner");
    }

    @Override
    public void jumpsInAir() {
        System.out.println("Dog and jump... are you kidding... always");
    }
```

Same way as you extend a class, you implement an interface.

Multiple interface may be implemented in one shot.

Default methods need not be overridden/ implemented. But you can do so if you want to

Interfaces overcome limitation of inheritance



Now this possible - Playful and Loveable are defined as interfaces, and Pet is an abstract class (not mandatory).