seed
beyond the obvious

Core Java

Interfaces

Objectives

- Identify the need for interfaces .
- Demonstrate Interfaces in detail using the Pet interface
- Distinguish between interfaces & abstract classes
- Distinguish between implements & extends keywords.

seed
beyond the obvious

Implementation Inheritance

- Where the sub class inherits all the code available in the super class.
- Useful for reusability of code.
- Problem arises when multiple inheritance has to be achieved .
- The OO language like C++ supports multiple inheritance at the cost of added complexity and ambiguity at times.
- Java does not support multiple inheritance.
- Java supports a single chain of implementation inheritance.

seed
beyond the obvious

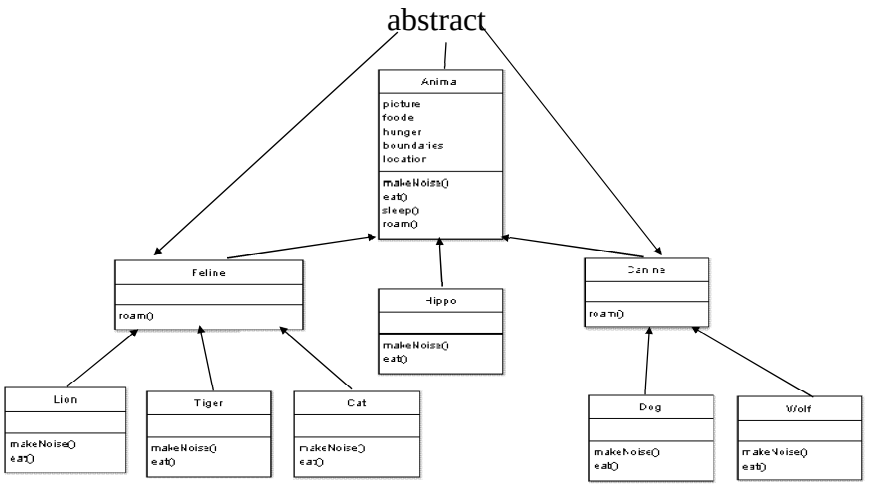
Interfaces - Their Need

- To overcome the lack of multiple inheritance Java uses INTERFACE INHERITANCE.
- Interface is essentially a collection of constants & abstract methods.
- The interface approach is sometimes known as programming by contract.

- But Why I need an interface???, So let's revisit our **Animal Simulation Program**

seed
beyond the obvious

Interface - Animal Simulation Program (ASP)



seed
beyond the obvious

Interface - Animal Simulation Program (ASP)

- Right now we can make use our **Animal Simulation Program** for a **science Fair Tutorial** on Animal objects
- What if we want to use it for a **PetShop** program???
- But we don't have any ***Pet*** behaviors. A pet needs methods like beFriendly() and play().

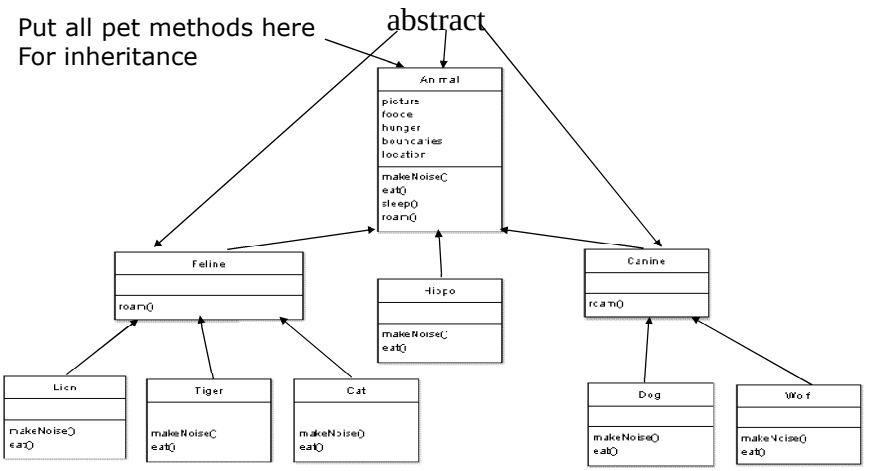
seed
beyond the obvious

Interface - Animal Simulation Program (ASP)

- Solution might be
  - Just add these methods to the **Dog** and **Cat** classes**,** by that we are not breaking anyone else's code, since we are not touching the existing methods that someone else's code might be calling on Dog and Cat objects
  - But this approach has a drawback. Can you guess what it is?
- Let's explore some design options for reusing some of our existing classes in a **PetShop** program

seed
beyond the obvious

Interface - ASP, Option 1



seed
beyond the obvious

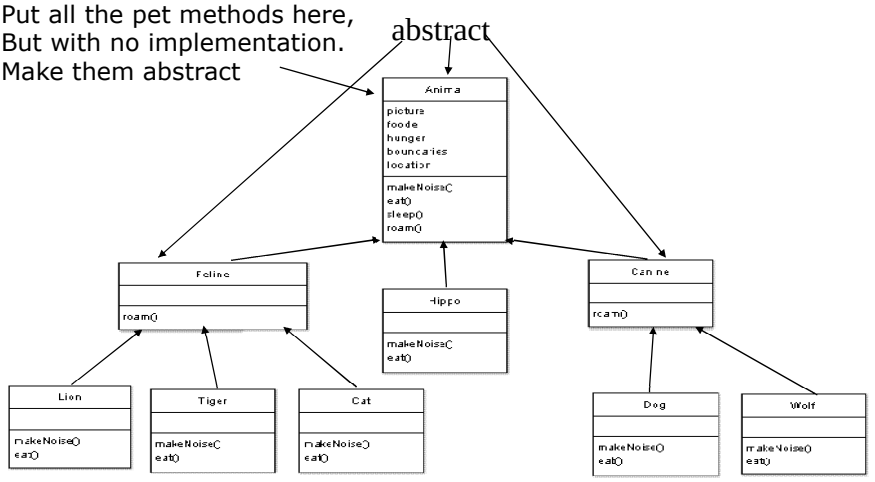
Interface - ASP, Option 1

- Pros
  - All the Animals will instantly inherit the pet behaviors. We won't have to touch the existing Animal subclasses at all, and any Animal subclasses created in future will also get to take the advantage of inheriting those methods. That way, class Animal can be used as the polymorphic type in any program that wants to treat the Animal as pets
- Cons
  - Can Lion, Tiger, Hippo be pets?
  - It could be dangerous to give non-pets pet methods.

seed
beyond the obvious

Interface - ASP, Option 2

Put all the pet methods here,
But with no implementation.
Make them abstract



seed
beyond the obvious

Interface ASP, Option 2

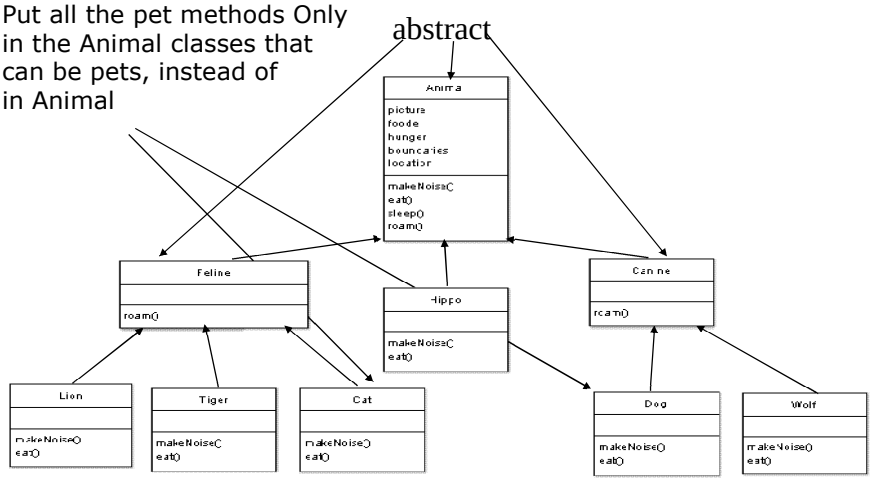
- Pros
  - That would give us all the benefits of Option 1, but without the drawback of having non-pet Animals running around with pet methods. All non pet animals must override the pet methods, but they can make the methods "do-nothings".
- Cons
  - The concrete Animal subclasses are forced to implement all the pet methods.
  - It's a waste of time to sit and type in each and every concrete non pet class, and all future subclasses also.

Interface ASP, Option 2

- Cons
  - While this does solve the problem of non-pets actually doing pet things the *contract is bad*. Every non-pet class would be announcing to the world that it, too has those pet methods, even though the methods wouldn't actually do anything when called.

Interface - ASP, Option 3

Put all the pet methods Only in the Animal classes that can be pets, instead of in Animal



seed
beyond the obvious

Interface - ASP, Option 3

- Pros
  - The methods are where they belong, and only where they belong
- Cons
  - Since it is not a contract, the compiler has no way to check you to see if you have implemented the methods correctly. Someone could easily come along to use pet Animal classes and find that not all of them work quite right

seed
beyond the obvious

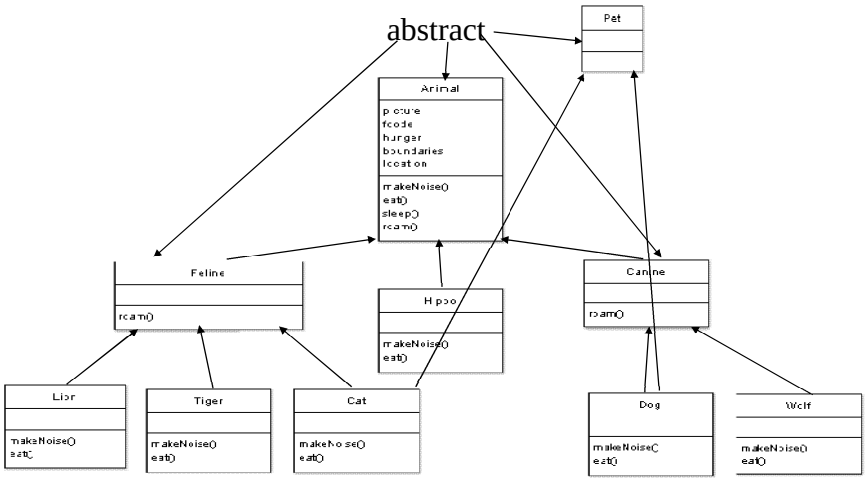
Interface - ASP, Option 3

- Cons
  - You can't use Animal as the polymorphic type now, because the compiler won't let you call a Pet method on an Animal reference because class Animal doesn't have the methods

Interface - Animal Simulation Program

- Look at our requirements closely, what we really need is
  - A way to have pet behavior in **just** the pet class
  - A way to guarantee that all pet classes have all of the same methods defined, without having to cross your fingers and hope all the programmers get it right
  - A way to take advantage of polymorphism so that all pets can have their pet methods called, without having to use arguments, return types, and arrays for each and every class
- It looks like we need two superclasses at the top

Interface - Animal Simulation Program



seed
beyond the obvious

Interface - Animal Simulation Program

- Interface to the rescue
  - Java gives you a solution through ***interface*** to the multiple inheritance problem by giving much of the polymorphic benefits of multiple inheritance without the pain and suffering from DDD.
  - The way the interface side-step the DDD is to **make all the methods abstract**, That way the subclass **must implement the** methods, so at runtime the JVM isn't confused about which of the two inherited versions it's supposed to call

seed
beyond the obvious

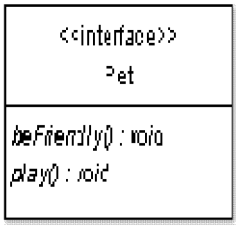
Interface - Animal Simulation Program

- **Syntax ->**

```
public interface Pet{
        void beFriendly();
        void play();
}
```

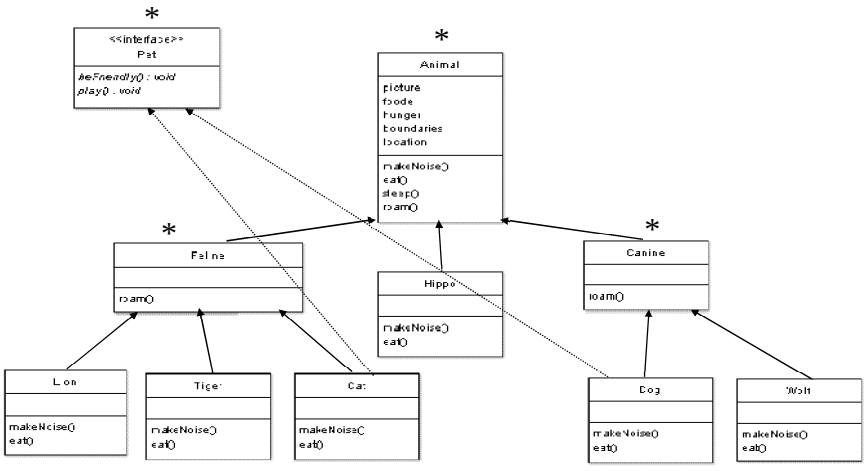
- All methods in an interface are by default abstract and public
- So now whoever want to be in a role of pet should implement a **Pet** interface
- It's a Dog and Cat only.
- **Syntax ->**

  public class Dog extends Canine implements Pet {…}
- Final Animal hierarchy now is….



seed
beyond the obvious

Interface - Animal Simulation Program



* Indicates abstract entities

seed
beyond the obvious

Interfaces

Interfaces are used for -

- Expanding the scope for polymorphism, across different class hierarchies, for example we can have a **RoboticDog** coming from **Robot** hierarchy to behave like a **Pet**
- Treating an object by the role it plays, rather than by the class type from which it was instantiated
- Implementing callback methods.

Example:-

Pet.java, Dog.java, Cat.java,
Animal.java, Feline.java, Canine.java

seed
beyond the obvious

Interfaces - Rules

- Methods in an interface are always public & abstract.
- Data members in an interface are always public, static & final.
- A sub class can only have a single superclass in java.
- But a class can implement any number of interfaces.

seed
beyond the obvious

Interface

The valid combinations are

- class extends class
- class implements interface
- interface extends interface

All other combinations are invalid.

seed
beyond the obvious

Interfaces & Abstract Classes

- Abstract classes are used only when there is a "is-a" type of relationship between the classes.
- You cannot extend more than one abstract class.
- Abstract class can contain abstract as well as implemented methods
- Interfaces can be implemented by classes that are not related to one another.
- You can implement more than one interface.
- Interfaces contain only abstract methods.

seed
beyond the obvious

Quick Recap

- Identified the need for interfaces .
- Demonstrated Interfaces in detail using the Pet interface
- Distinguished between interfaces & abstract classes
- Distinguished between implements & extends keywords.

seed
beyond the obvious