Java & OOP

3. Using Classes

Use first

- Learn to use them
- Different ways of using
- How parts of the classes are used

Kid class

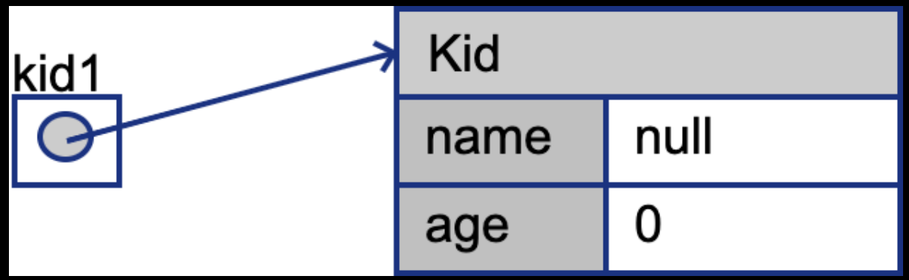
- Has 2 properties (name, age)
- A Behaviour to respond to a question "who the kid is"

Creating a Kid

- Construct a Kid object
- Set values for properties
- Enquire who that kid is

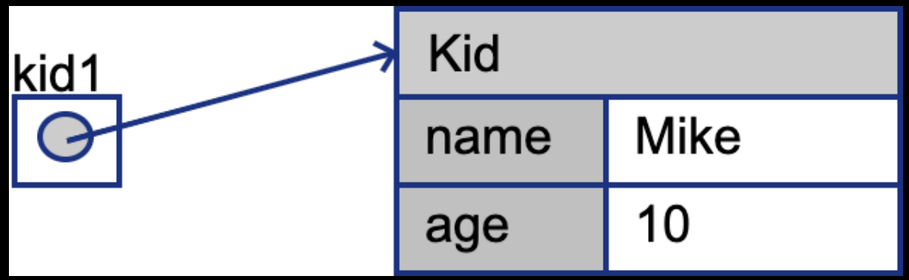
Creating a Kid

```
public class Main {

    public static void main(String[] args) {
        Kid kid1 = new Kid();
        kid1.name = "Mike";
        kid1.age = 10;
        String info1 = kid1.whoAreYou();
        System.out.println(info1);
    }
}
```



Creating a Kid

```
public class Main {

    public static void main(String[] args) {
        Kid kid1 = new Kid();
        kid1.name = "Mike";
        kid1.age = 10;
        String info1 = kid1.whoAreYou();
        System.out.println(info1);
    }
}
```



I am Mike and I am 10 years old!

Try this

```
public class Main {

    public static void main(String[] args) {
        Kid kid1 = new Kid();
        String info1 = kid1.whoAreYou(); //Moved up
        kid1.name = "Mike";
        kid1.age = 10;
        System.out.println(info1);
    }
}
```

I am null and I am 0 years old!

What happened?

- Behaviour invoked before proper initialization

Enforce proper intialization!

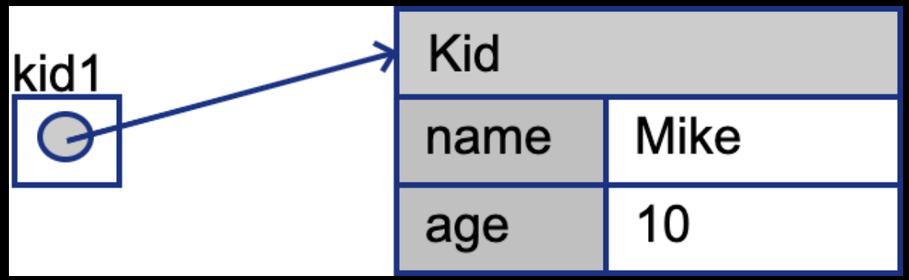
Creating Kid

- Let us create a second `Kid`

```
Kid kid2 = new Kid("Dave", 8);
String info2 = kid2.whoAreYou();
System.out.println(info2);
```

I am Dave and I am 8 years old!

- Good enough!



Dog class

- Let us create Dog objects and hand them over to the kids
- Dog has 2 properties - name and color
- A behaviour - play

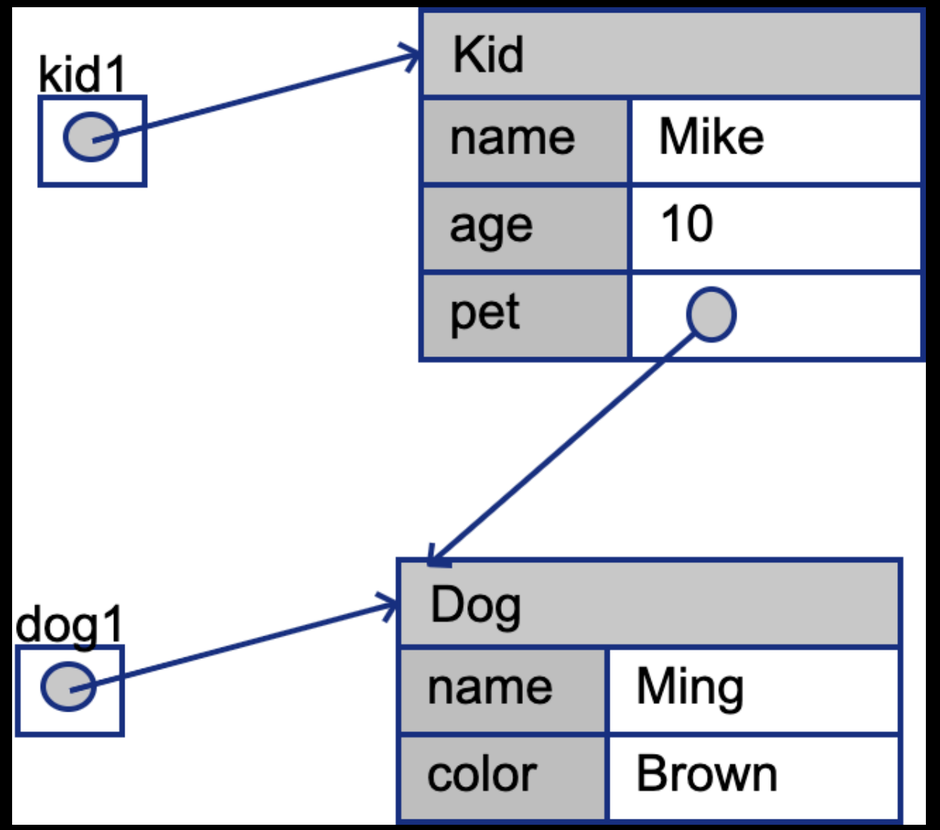
Dependency (a.k.a Association)

- Kid is going to invoke `play` behaviour on Dog
- Kid is a client
- Kid depends on Dog
- How will a Kid object knows about a Dog object?
  - Kid has a property by name `pet` which is `Dog` type
- A Dog is associated with a Kid as pet

Play with the pets

```
Dog dog1 = new Dog("Ming", "Brown");
Dog dog2 = new Dog("Dancer", "white");

//Directly assign
kid1.pet = dog1;
//do it through a behaviour
kid2.setPet(dog2);

kid1.play();
kid2.play();
```

- Output
  Mike playing with Ming
  Ming is running...
  Dave playing with Dancer
  Dancer is running...



Association Style

- Property assignment
  - Kid exposes the pet property variable
  - To be avoided as there is no scope for validation
- Using a behaviour (function) to set property
  - Kid need not expose the variable `pet`
  - Useful to add validation in Kid's behaviour while accepting the pet
- This is permenant

Association - Temporary (Just in time)

- `Kid` has a behaviour to `playWith` any dog
- Here `Kid` will know this dog only for the execution time of `playWith`

```
kid2.playWith(dog1);
```

- Output
  Dave playing with Ming
  Ming is running...

Association - Safe way

- Setting a property after constructing Kid - We might forget
- This association is long and lifetime of the kid Class
- What if someone invokes `play` without setting a `pet`?
- Since `pet` is a property why dont we pass it while creating a `Kid` object?

```Java
Dog dog3 = new Dog("Tiger", "Gold");
Kid kid3 = new Kid("Thea", 9, dog3);
kid3.play();
```

Association - Injection

- For the system to work we associate a dependency into a client
- Also called as "Injection"
- Through a function
  - These functions are normally of the form `set<Property>(value)`
  - Setter Injection

Association - Injection

- While creating the client
  - This is done using ***constructor***
  - Constructor Injection
- Temporary association
  - Parameter to a function
  - Parameter Injection
- Spring is a framework used for injecting dependency automatically

Interface & Encapsulation

- We used behaviour mostly rather than working with properties directly
- A Class exposes behaviour and not properties directly
- Property is part of implementation
- Publicly exposed behaviour - ***Interface***
- Hiding implementation behind an interface - ***Encapsulation***

Next

Writing Classes