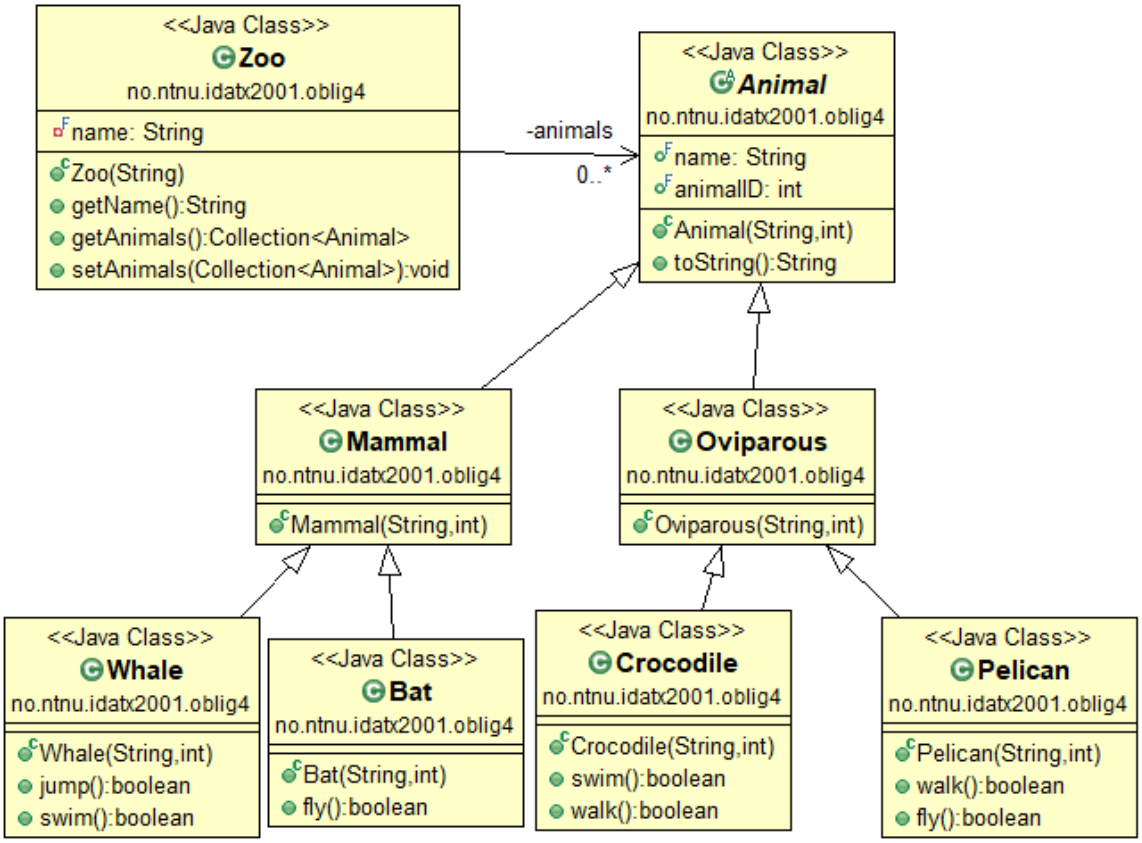
# Grensesnitt og unntakshåndtering

Du skal nå jobbe med å bygge en applikasjon for en dyrepark. Vi ser nærmere på de forskjellige klassene og har spesielt fokus på implementasjon av grensesnitt og unntakshåndtering i denne delen av øvingen. Gitt følgende klasse-hierarki:



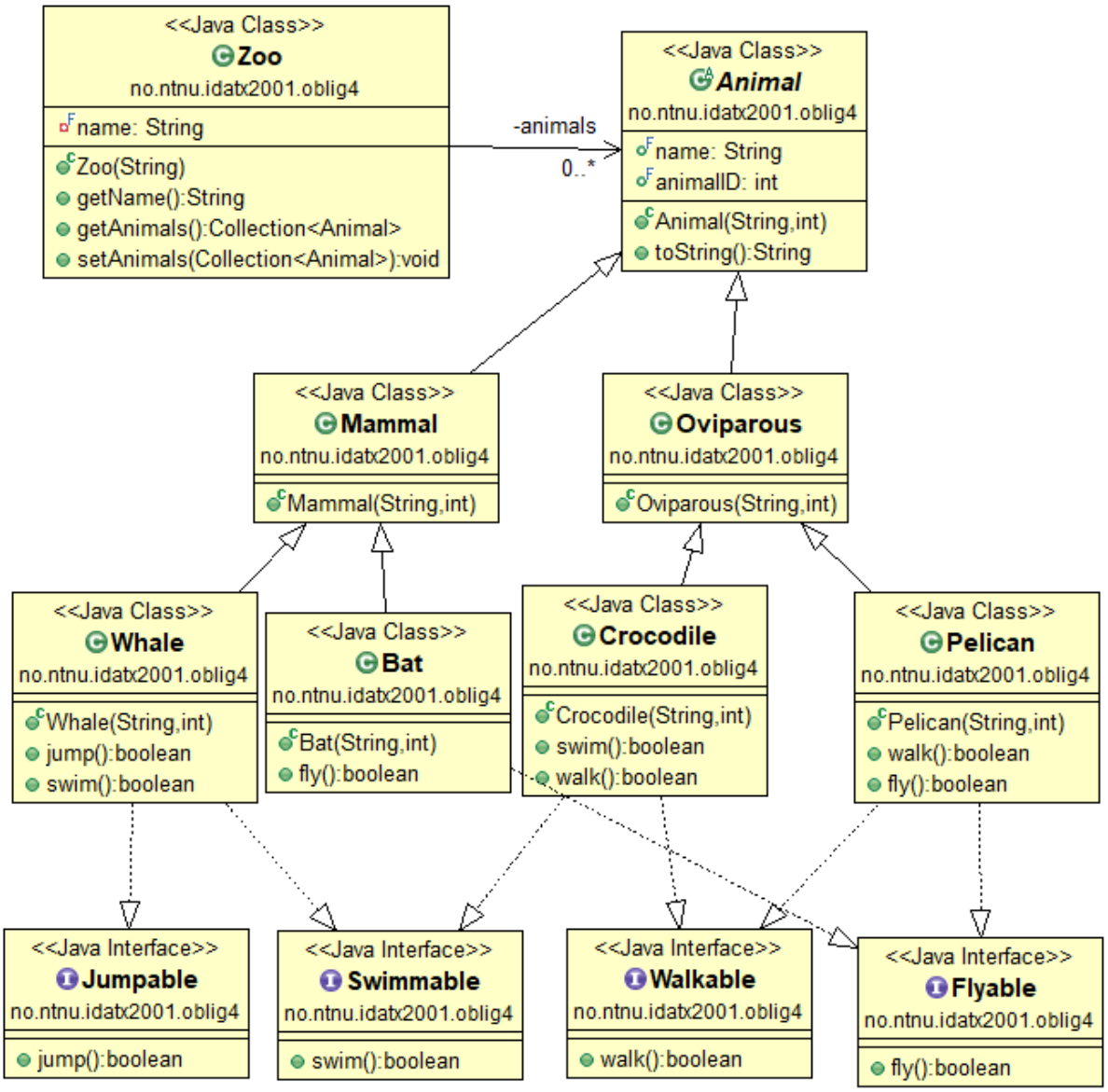

**Figur 1: Klassediagram for dyrepark**

Klassen Zoo (Se vedlegg) inneholder en samling med forskjellige dyr (Animal). Hver dyr registreres med navn og unik id (name og animalID). Videre er dyrene kategorisert i pattedyr (Mammal) og eggleggende (Oviparous). Videre har vi i denne oppgaven definert to typer pattedyr: hval (Whale) og flaggermus (Bat). Av eggleggende dyr har vi definert krokodille (Crocodile) og pelikan (Pelican).

Klassene Animal, Mammal og Oviparous er allerede definert (se vedlegg).

## Oppgave 1: Definering av grensesnitt klassene

I klasse-diagrammet nedenfor har vi definert følgende grensesnitt-klasser: Jumpable, Swimmable, Walkable og Flyable:




**Figur 2: Klassediagram med grensesnitt.**

Dyr som implementerer Jumpable kan hoppe. I dette tilfellet er det kun hvaler som kan implementere Jumpable-grensesnitt.

Dyr som implementerer Swimmable kan svømme. I dette tilfellet er det både hvaler og krokodiller som kan implementere Swimmable-grensesnitt.

Dyr som implementerer Walkable kan gå. I dette tilfellet er det både pelikaner og krokodiller som kan implementere Walkable-grensesnitt.

Dyr som implementerer Flyable kan fly. I dette tilfellet er det både pelikaner og flaggermus som kan implementere Flyable-grensesnitt.

Skriv java-kode for grensesnitt- og de klassene som implementerer disse klassene (Whale, Bat, Crocodile, Pelican).

Jumpable har metoden jump(), Swimmable har metoden swim(), Walkable har metoden walk() og Flyable har metoden fly(). Grensesnittene skal implementeres av dyre-klassene. Se figur 2.

## Oppgave 2: Bruk grensesnitt til å filtrere på forskjellige type dyr

Gitt følgende klient klasse:

```
package no.ntnu.idatx2001.oblig4;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

public class ZooClient {

	public static void main(String[] args) {

		Zoo zoo = new Zoo("Kristiansand Dyrepark");


		Collection<Animal> animals = new ArrayList<Animal>();

		animals.add(new Crocodile("Crocodylus niloticus", 1001));
		animals.add(new Crocodile("Crocodylus niloticus", 1002));
		animals.add(new Crocodile("Crocodylus porosus", 1101));
		animals.add(new Crocodile("Crocodylus porosus", 1102));

		animals.add(new Pelican("Brown Pelican  ", 4001));
		animals.add(new Pelican("Dalmatian Pelican  ", 4101));


		animals.add(new Whale("Blue whale", 2001));
		animals.add(new Whale("Blue whale", 2002));
		animals.add(new Whale("Minke whale", 2101));
		animals.add(new Whale("Minke whale", 2102));

		animals.add(new Bat("Acerodon ", 3001));
		animals.add(new Bat("Cistugo  ", 3002));
```

```
            zoo.setAnimals(animals);

            …

        }
}
```

a) Lag en lambda uttrykk som finner alle dyr som kan **fly**. Kall deretter metoden fly() på hvert objekt.
b) Lag en lambda uttrykk som finner alle **pattedyr** som kan **hoppe**. Kall deretter metoden jump() på hvert objekt.

## Oppgave 3: Unntakshåndtering

Uttrykket nedenfor kompilerer fint, men feiler under kjøring.

```
List<Object> walker = zoo.getAnimals().stream().filter(p -> p instanceof Walker
).collect(Collectors.toList());

walker.stream().forEach(p -> {
        ((Flyable)p).fly();
  }
);
```

a) Bruk try-catch blokk å fange riktig exception slik at programmet ikke feiler.
b) Endre i koden slik at programmet stopper helt når exception oppstår.
c) Lag en egen exception klasse (ZooException) som kastes når når i applikasjonen går galt.

# Vedlegg

```java
package no.ntnu.idatx2001.oblig4;

import java.util.ArrayList;
import java.util.Collection;


public class Zoo {
	private final String name;

	private Collection<Animal> animals = new ArrayList<Animal>();

	public Zoo(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public Collection<Animal> getAnimals() {
		return animals;
	}

	public void setAnimals(Collection<Animal> animals) {
		this.animals.addAll(animals);
	}
}


public abstract class Animal {
	public final String name;
	public final int animalID;

	public Animal(String name, int code) {
		this.name = name;
		this.animalID = code;
	}

	@Override
	public String toString() {
		return "Animal [name=" + name + ", code=" + animalID + "]";
	}
}


public class Mammal extends Animal {

	public Mammal(String name, int code) {
		super(name, code);
```

```
		// TODO Auto-generated constructor stub
	}

}

public class Oviparous extends Animal {

	public Oviparous(String name, int code) {
		super(name, code);
		// TODO Auto-generated constructor stub
	}
}
```