Iteration 1

```java
import java.util.Random;

class Main {
    public static void main(String[] args) {
        Random rng = new Random();
        double liveChance = rng.nextDouble();
        if (liveChance > 0.5) {
            System.out.println("Cat lives");
        } else {
            System.out.println("Cat dies");
        }
    }
}
```

Iteration 2

```java
import java.util.Random;

class Cat {
    private final String name;

    Cat(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Cat: " + name;
    }
}

class Box {
    private final Cat cat;
    private static final Random rng = new Random();

    Box(Cat cat) {
        this.cat = cat;
    }

    public Box experiment() {
        double liveChance = rng.nextDouble();
        if (liveChance > 0.5) {
            return this;
        }
        return new Box(null);
    }

    private boolean isEmpty() {
        return cat == null;
    }

    public String toString() {
        if (!isEmpty()) {
            return String.format("Box<%s>", cat.toString());
        }
        return "Box<Dead Cat>";
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Cat cat = new Cat("Mittens");
        Box box = new Box(cat);
        box = box.experiment();
        System.out.println(box);
    }
}
```

Iteration 3

```java
import java.util.Random;

record Cat(String name) {
    @Override
    public String toString() {
        return "Cat: " + name;
    }
}

class Box<T> {
    private final T thing;
    private static final Random RNG = new Random();

    public Box(T thing) {
        this.thing = thing;
    }

    public Box<T> experiment() {
        double liveChance = RNG.nextDouble();
        if (liveChance > 0.5) {
            return this;
        }
        return new Box<T>(null);
    }

    private boolean isEmpty() {
        return thing == null;
    }

    public String toString() {
        if (!isEmpty()) {
            return String.format("Box<%s>", thing);
        }
        return "Box.empty";
    }
}

class Main {
    public static void main(String[] args) {
        Box<Cat> box = new Box<Cat>(new Cat("Mittens"))
            .experiment();
        System.out.println(box);
    }
}
```

Iteration 4

```java
import java.util.Random;

interface Converter<T, R> {
    R convert(T thing);
}

record Cat(String name) {
    @Override
    public String toString() {
        return "Cat: " + name;
    }
}

class Box<T> {
    private final T thing;

    private Box(T thing) {
        this.thing = thing;
    }

    public static <T> Box<T> of(T thing) {
        if (thing == null) {
            return Box.<T>empty();
        }
        return new Box<T>(thing);
    }

    public static <T> Box<T> empty() {
        return new Box<T>(null);
    }

    public <R> Box<R> convert(Converter<T, R> converter) {
        if (!isEmpty()) {
            return Box.<R>of(converter.convert(thing));
        }
        return Box.<R>empty();
    }

    private boolean isEmpty() {
        return thing == null;
    }

    public String toString() {
        if (!isEmpty()) {
            return String.format("Box<%s>", thing);
        }
        return "Box.empty";
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Random RNG = new Random();
        double liveChance = RNG.nextDouble();

        Converter<Cat, Cat> experiment = x -> liveChance > 0.5
            ? x
            : null;

        Box<String> box = Box.<Cat>of(new Cat("Mittens"))
            .convert(experiment)
            .convert(x -> String.format("%s survived!", x.name()));
        System.out.println(box);
    }
}
```

Iteration 5

```
import java.util.Random;

interface Converter<T, R> {
    R convert(T thing);
}

interface Tester<T> {
    boolean test(T thing);
}

record Cat(String name) {
    @Override
    public String toString() {
        return "Cat: " + name;
    }
}

class Box<T> {
    private final T thing;

    private Box(T thing) {
        this.thing = thing;
    }

    public static <T> Box<T> of(T thing) {
        return thing == null ? Box.<T>empty() : new Box<T>(thing);
    }

    public static <T>Box<T> empty() {
        return new Box<T>(null);
    }

    public <R> Box<R> convert(Converter<T, R> converter) {
        return !isEmpty() ? Box.<R>of(converter.convert(thing)) : Box.<R>empty();
    }

    public Box<T> test(Tester<T> tester) {
        return !isEmpty() && tester.test(thing) ? this : Box.<T>empty();
    }

    private boolean isEmpty() {
        return thing == null;
    }

    public String toString() {
        return !isEmpty() ? String.format("Box<%s>", thing) : "Box.empty";
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Random RNG = new Random();
        double liveChance = RNG.nextDouble();

        Tester<Cat> experiment = x -> liveChance > 0.5;

         Box<Cat> box = Box.<Cat>of(new Cat("Mittens"))
            .test(experiment) ;
        System.out.println(box);
    }
}
```

Iteration 6

```java
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.Random;

record Cat(String name) {
    @Override
    public String toString() {
        return "Cat: " + name;
    }
};

class Box<T> {
    private final T thing;

    private Box(T thing) {
        this.thing = thing;
    }

    public static <T> Box<T> of(T thing) {
        return thing == null ? Box.<T>empty() : new Box<T>(thing);
    }

    public static <T>Box<T> empty() {
        return new Box<T>(null);
    }

    public <R> Box<R> map(Function<? super T, ? extends R> converter) {
        return !isEmpty() ? Box.<R>of(converter.apply(thing)) : Box.<R>empty();
    }

    public Box<T> filter(Predicate<? super T> tester) {
        return !isEmpty() && tester.test(thing) ? this : Box.<T>empty();
    }

    private boolean isEmpty() {
        return thing == null;
    }

    public String toString() {
        return !isEmpty() ? String.format("Box<%s>", thing) : "Box.empty";
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Random RNG = new Random();
        double liveChance = RNG.nextDouble();

        Box<Cat> box = Box.<Cat>of(new Cat("Mittens"))
            .filter(x -> liveChance > 0.5) ;
        System.out.println(box);
    }
}
```

Iteration 7

```
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.Random;

record Cat(String name) {
    @Override
    public String toString() {
        return "Cat: " + name;
    }
};

class Box<T> {
    private final T thing;

    private Box(T thing) {
        this.thing = thing;
    }

    public static <T> Box<T> of(T thing) {
        return thing == null ? Box.<T>empty() : new Box<T>(thing);
    }

    public static <T>Box<T> empty() {
        return new Box<T>(null);
    }

    public <R> Box<R> map(Function<? super T, ? extends R> converter) {
        return !isEmpty() ? Box.<R>of(converter.apply(thing)) : Box.<R>empty();
    }

    public Box<T> filter(Predicate<? super T> tester) {
        return !isEmpty() && tester.test(thing) ? this : Box.<T>empty();
    }

    public <R> Box<R> flatMap(Function<? super T, ? extends Box<? extends R>> flatMapper) {
        if (!isEmpty()) {
            Box<? extends R> flatMapResult = flatMapper.apply(thing);
            return Box.<R>of(flatMapResult.thing);
        }
        return Box.<R>empty();
    }

    private boolean isEmpty() {
        return thing == null;
    }

    public String toString() {
        return !isEmpty() ? String.format("Box<%s>", thing) : "Box.empty";
    }
}
```

```java
class Main {
    public static void main(String[] args) {
        Random RNG = new Random();

        Box<Cat> boxOne = Box.<Cat>of(new Cat("Mittens"))
            .filter(x -> RNG.nextDouble() > 0.5);

        Box<Cat> boxTwo = Box.<Cat>of(new Cat("Garfield"))
            .filter(x -> RNG.nextDouble() > 0.5);

        Box<String> combinedBox = boxOne.flatMap(x ->
                boxTwo.map(y -> y.toString() + ", " + x.toString()));

        System.out.println(combinedBox);
    }
}
```

Iteration 8

```java
import java.util.Random;
import java.util.function.Function;
import java.util.function.Predicate;

record Cat(String name) {
    @Override
    public String toString() {
        return "Cat: " + name;
    }
};
```

```
interface Box<T> {
    public <R> Box<R> map(Function<? super T, ? extends R> mapper);
    public <R> Box<R> flatMap(Function<? super T, ? extends Box<? extends R>> flatMapper);
    public Box<T> filter(Predicate<? super T> tester);

    public static <T> Box<T> of(T thing) {
        return thing == null ? Box.<T>empty() : new Box<T>() {
            public <R> Box<R> map(Function<? super T, ? extends R> mapper) {
                return Box.<R>of(mapper.apply(thing));
            }

            public <R> Box<R> flatMap(Function<? super T, ? extends Box<? extends R>>
flatMapper) {
                Box<? extends R> result = flatMapper.apply(thing);
                return result.<R>map(x -> x);
            }

            public Box<T> filter(Predicate<? super T> tester) {
                return tester.test(thing) ? this : Box.<T>empty();
            }

            public String toString() {
                return String.format("Box<%s>", thing);
            }
        };
    }

    public static <T> Box<T> empty() {
        return new Box<T>() {
            public <R> Box<R> map(Function<? super T, ? extends R> mapper) {
                return Box.<R>empty();
            }

            public <R> Box<R> flatMap(Function<? super T, ? extends Box<? extends R>>
flatMapper) {
                return Box.<R>empty();
            }

            public Box<T> filter(Predicate<? super T> tester) {
                return Box.<T>empty();
            }

            public String toString() {
                return "Box.empty";
            }
        };
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Random RNG = new Random();
        double liveChance = RNG.nextDouble();

        Box<Cat> box = Box.<Cat>of(new Cat("Mittens"))
            .filter(x -> liveChance > 0.5) ;
        System.out.println(box);
    }
}
```