Practice Question 1

For this question, consider the following class.

```
public class Cat {
    public static String LATEST_KITTEN_NAME;
    private String name;
    private int color;

    public Cat(String name) {
        this.name = name;
        this.color = 0x808080; // Default to grey
        LATEST_KITTEN_NAME = this.name;
    }

    public Cat(String name, int color) {
        this.name = name;
        this.color = color;
        LATEST_KITTEN_NAME = this.name;
    }

    public String getName() {
        return this.name;
    }

    public int getColor() {
        return this.color;
    }

    public void rename(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        // The 0x%06X will format a 0-padded hexadecimal value of length 6
        // e.g., if color = 0xF0, then it will format to "0x0000F0"
        return String.format("A 0x%06X colored cat named %s", color, name);
    }
}
```

Part 1

Explain whether each of the following independent code blocks will execute or fail to compile. Assume they are written in a file separate from the Cat class. If the code executes, describe the output or return value. If the code fails to compile or run, explain why.

a)
```
Cat cat = new Cat("Mochi");
System.out.println(Cat.getName());
```
a) Does not compile.
Tries to access an instance variable statically.

b)
```
Cat cat = new Cat("Mochi");
System.out.println(Cat.LATEST_KITTEN_NAME);
```
b) Compiles and outputs "Mochi"

c)
```
Cat catA = new Cat("Mochi");
Cat catB = new Cat("Harvest");
System.out.println(Cat.LATEST_KITTEN_NAME);
```
c) Compiles and outputs "Harvest"

d)
```
System.out.println(Cat.LATEST_KITTEN_NAME);
```
d) Compiles and outputs "null"

e)
```
Cat cat = new Cat("Mochi");
Cat duplicat = cat;
duplicat.rename("Momo");
System.out.println(cat);
```
e) Compiles and outputs"A 0x808080 colored cat named Momo"

Part 2

For this part, consider the following code:

```
Cat a = new Cat("Mochi");
Cat b = new Cat("Harvest", 0xff8000);
Cat c = new Cat("Mochi", 0x808080);
Cat d = a;
```

What will each of the following code fragments print or evaluate to? Assume each is **independent** of the other.

f) `a == b` False

g) `a == c` False

h) `a == d` True

i)
```
d.rename("Momo")
System.out.println(a == d)
```
True

j)
```
a = b
System.out.println(a == d)
```
False

Part 3

Write a standard `equals()` method for the `Cat` class. An object is considered equal to a `Cat` object if it can be cast as one and has equivalent instance variable values.

```
public boolean equals(Object other) {
    if (other instanceof Cat cat) {
        return color == cat.color && Objects.equals(name, cat.name);
    }
    return false;
}
```

Part 4

For this part, consider the following code:

```
Cat a = new Cat("Mochi");
Cat b = new Cat("Harvest", 0xff8000);
Cat c = new Cat("Mochi", 0x808080);
Cat d = a;
```

Assume your equals method from part 3 is correct. What will each of the following code fragments print or evaluate to? Assume each is **independent** of the other.

k) `a == c` false

l) `a.equals(c)` true

m) `a == d` true

n)
```
d.rename("Momo")
a.equals(d)
```
true

o)
```
a = null
a.equals(d)
```
NullPointerException occurs

Practice Question 2

Consider the following program:

```
public class Exceptional {
    public static void main(String[] args) {
        System.out.print("1");
        try {
            // Insert function call here!
        } catch (NullPointerException | ArithmeticException e) {
            System.out.print("2");
        } catch (IllegalArgumentException e) {
            System.out.print("3");
            throw e;
        } finally {
            System.out.print("4");
        }
        System.out.print("5");
    }
    public static void bad() {
        throw new NullPointerException();
    }
    public static void oops() {
        throw new ArithmeticException();
    }
    public static void criminal() {
        throw new IllegalArgumentException();
    }
    public static void good() {}
}
```

For each of the following function calls, what would the program print if it was inserted in the commented line within `main`? Indicate if the program would print a stack trace.

Call	**Output**	**Stack Trace? Y/N**
`bad()`	1245	N
`oops()`	1245	N
`criminal()`	134	Y
`good()`	145	N

Practice Question 3

Consider the game of *UNO* described by this quote from wikipedia:

> *Uno, stylized as UNO, is a proprietary American shedding-type card game originally developed in 1971 by Merle Robbins in Reading, Ohio, a suburb of Cincinnati, that housed International Games Inc., a gaming company acquired by Mattel on January 23, 1992.*
>
> *Played with a specially printed deck, the game is derived from the crazy eights family of card games which, in turn, is based on the traditional German game of mau-mau.*

A standard game of *UNO* consists of 108 cards as follows:

- 19 Blue Numbered cards: one 0 and two of each 1 - 9
- 19 Green Numbered cards: one 0 and two of each 1 - 9
- 19 Red Numbered cards: one 0 and two of each 1 - 9
- 19 Yellow Numbered cards: one 0 and two of each 1 - 9
- 8 Draw Two cards: 2 each in blue, green, red, and yellow
- 8 Reverse cards: 2 each in blue, green, red, and yellow
- 8 Skip cards: 2 each in blue, green, red, and yellow
- 4 Wild cards
- 4 Wild Draw Four cards

As shown above, cards have different types and each type might have specific data associated with it. Additionally, each card in Uno can be *played*. When played, numbered, "draw two", "reverse", "skip", "wild", and "Wild Draw Four" cards each perform unique effects.

A card can be considered *playable* based on the current state of the game. More specifically, a card is determined playable based on another card that was previously played (i.e., playability is a function of the state of another card).

Draw the type hierarchy you would use to implement this behavior in Java. This should include type names, method signatures, instance variable names and types, and if applicable, static variable names and types.

For each type in the hierarchy, be clear about whether it is a regular class, an abstract class, or an interface.

For each method name, be clear about whether it is a regular method, an abstract method, or a default method. If a method is overridden from a super type, that should be apparent from your figure.