1 It's a Bird! It's a Plane! It's a CatBus!

On a research expedition studying air traffic, we discovered a new species: the Flying Interfacing CatBus, which acts like a vehicle and has the ability to honk (safety is important!).

(a) Given the Vehicle and Honker interfaces, fill out the CatBus class so that CatBuses can rev their engines and honk at other CatBuses with a CatBus-specific honk.

```java
interface Vehicle {
    public void revEngine();
}

interface Honker {
    public void honk();
}

public class CatBus _____ _____, _____ {
    @Override
    _____ _____ _____ { /* CatBus revs engine, implementation hidden */ }

    @Override
    _____ _____ _____ { /* CatBus honks, implementation hidden */ }

    /** Allows CatBus to honk at other CatBuses. */
    public void conversation(CatBus target) {
        honk();
        target.honk();
    }
}
```

Solution:

```
1   interface Vehicle {
2       public void revEngine();
3   }
4
5   interface Honker {
6       public void honk();
7   }
8
9   public class CatBus implements Vehicle, Honker {
10      @Override
11      public void revEngine() {
12          // CatBus revs its engine, implementation not shown
13      }
14
15      @Override
16      public void honk() {
17          // CatBus honks, implementation not shown
18      }
19
20      /** Allows CatBus to honk at other CatBuses. */
21      public void conversation(CatBus target) {
22          honk();
23          target.honk();
24      }
25  }
```

(b) It's a lovely morning in the skies and we've encountered a horrible `Goose`, which also `implements Honker` (it has a knife in its beak!). Modify the `conversation` method signature so that `CatBus`es can `honk` at both `CatBus` *and* `Goose` objects while only having one argument, `target`.

Solution:

We can change the method signature so that the type of the parameter `target` is `Honker` (both `CatBus` and `Goose` implement `Honker`):

```
/** Allows CatBus to honk at other both CatBuses and Gooses. */
public void conversation(Honker target) {
    honk();
    target.honk();
}
```

(c) Assume that we have another class, `CanadaGoose`, which extends `Goose`. Which of the following lines compile?

Solution:

```
Honker cb = new CatBus(); // Compiles - a CatBus is a kind of Honker
CatBus g = new Goose(); // Errors - a Goose is not a CatBus, even though they are both Honkers
                        // ("siblings" in the inheritance tree)
Honker h = new Honker(); // Errors - cannot instantiate an interface
CanadaGoose cg = new Goose(); // Errors - a CanadaGoose is a Goose,
                              // but not necessarily the other way around
Honker hcg = new CanadaGoose(); // Compiles - a CanadaGoose is a kind of Honker
```

2 Raining Cats and Dogs

(a) What would Java do after executing the main method in the TestAnimal class? Fill in the table provided with the method saved at compile time, the method called at runtime, and overall output for lines 8-19 if applicable. If there is an error, write whether it is a runtime error or compile time error, and then proceed through the rest of the code as if the erroneous line were not there.

```java
public class Animal {
    public String name, noise;

    public Animal(String name) {
        this.name = name;
        this.noise = "Huh?";
    }

    public void greet(Animal a) { System.out.println("Hi " + a.name + ", I'm " + name); }
    public void play() { System.out.println("I love to play! " + noise); }
    public static void sleep() { System.out.println("Naptime!"); }
}

public class Cat extends Animal {
    public Cat(String name) {
        super(name);
        this.noise = "Meow!";
    }

    public void greet(Animal a) { System.out.println("Cat " + name + " says: " + noise); }
    public void play() {
        System.out.println("Woo it is so much fun being a cat! " + noise);
    }
}
public class Dog extends Animal {
    public Dog(String name) {
        super(name);
        noise = "Woof!";
    }

    public void greet(Animal a) { System.out.println("Dog " + name + " says: " + noise); }
    public void play() {
        System.out.println("Woo it is so much fun being a dog! " + noise);
    }
    public static void sleep() { System.out.println("I love napping!"); }
}
```

```
1    public class TestAnimal {
2        public static void main(String[] args) {
3            Animal a = new Dog("Pluto");
4            Animal b = new Animal("Bear");
5            Cat c = new Cat("Garfield");
6            Dog d = new Dog("Lucky");
7
8            Cat e = new Animal("Kitty");
9            a.greet(c);
10           a.sleep();
11           c.play();
12           c.greet(d);
13           ((Animal) c).greet(d);
14           d.sleep();
15           a = c;
16           a.play(14);
17           ((Cat) b).play();
18           d = (Dog) a;
19           c = a;
20       }
21   }
```

Solution:

line	Compile time (static)	Runtime (dynamic)	Output
8	Error: an Animal is not necessarily a Cat	N/A	Compiler error
9	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
10	Animal's sleep()	N/A - sleep() is static!	"Naptime!"
11	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
12	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
13	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
14	Dog's sleep()	N/A - sleep() is static!	"I love napping!"
15	works because a Cat is-an Animal	works because a Cat is-an Animal	ok - nothing is printed
16	Error: Animal does not define play(int)	N/A	Compiler error
17	Animal's play() (cast works here because Animal could be a Cat)	Error: an Animal is not necessarily a Cat	Runtime error
18	Works because Animal could be a Dog	Error: a Cat is not a Dog (can't cast between siblings)	Runtime error
19	Error: c is static type Cat but a is static type Animal	N/A	Compiler error

(b) Spoiler alert! There is an error on the last line, line 19. How could we fix this error?

Solution:

The compilation error on line 19 is because we are trying set `c`, which is of static type `Cat` to be equal to `a`, when the static type of `a` is `Animal`. Even though at runtime, `a` really does have dynamic type `Cat`, the compiler only sees static types so it doesn't believe that this assignment is valid. The compiler only sees that we are trying to set a `Cat` variable to point to an `Animal`, and an `Animal` isn't a `Cat`!

We could fix this error by casting a to be a Cat, such that the line reads `c = (Cat) a;`. This would be a valid cast, as the compiler agrees that a variable of static type `Animal` could potentially hold a `Cat`, and so our request is feasible. Because the cast works, then the assignment is also now valid because a variable of static type `Cat` can be told to point to the same thing as another variable of (temporary) static type `Cat`. At runtime, this line will be fine because we were telling the truth: `a` really is a `Cat` dynamically!