Inheritance

Discussion 03

Example Agenda

- 1:10 - 1:15 ~ announcements
- 1:15 - 1:30 ~ content review
- 1:30 - 1:40 ~ question 1
- 1:40 - 1:55 ~ question 2
- Question 3 if time

Announcements

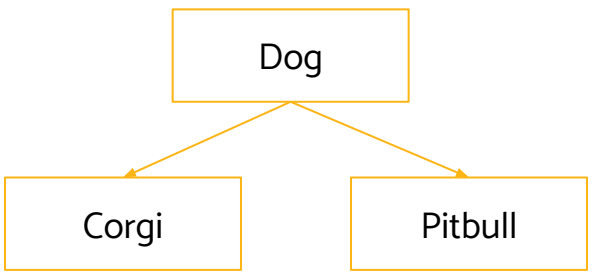
- Midterm 1 on Thursday 2/15 7-9 PM
    - Review Session Friday 2/9 11-1PM in Soda labs
- No lab assignment this week (Project 1 Workday)
- Project 1A due this Monday 2/5
- Project 1B due next Monday 2/12
- Project 1C due Tuesday 2/20
- Weekly Survey 3 due this Monday 2/5

Content Review

Classes

Subclasses (or child classes) are classes that inherit from another class. This means that they have access to all non-private functions and variables of their parent class in addition to any functions and variables defined in the child class.
Example: Corgi, Pitbull

Superclasses or parent classes are classes that are inherited by another class.
Example: Dog



Fun with Methods

Method Overloading is done when there are multiple methods with the same name, but different parameters.

```
public void barkAt(Dog d) { System.out.print("Woof, it's another dog!"); }
public void barkAt(CS61BStaff s) { System.out.print("Woof, what is this?"); }
```

* Food for thought: what is an advantage of method overloading? Hint: think about `System.out.print`

Method Overriding is done when a subclass has a method with the exact same function signature as a method in its superclass. It is usually marked with the `@Override` tag.

In Dog class:

```
public void speak() { System.out.print("Woof, I'm a dog!"); }
```

In Corgi Class, which inherits from Dog:

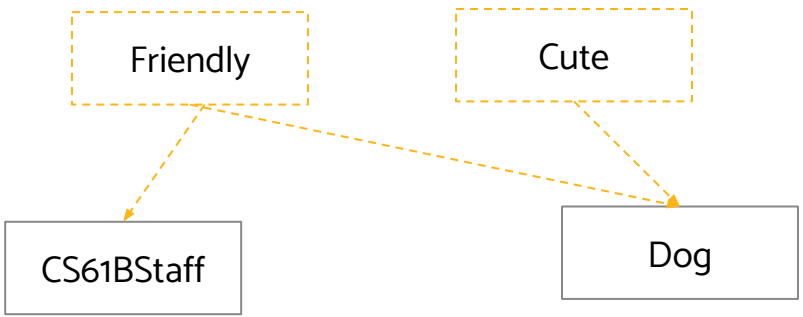
```
@Override
public void speak() { System.out.print("Woof, I'm a corgi!"); }
```

Interfaces

Interfaces are implemented by classes. They describe a narrow ability that can apply to many classes that may or may not be related to one another.

They do not usually implement the methods they specify, but can do so with the `default` keyword. Interface methods are inherently `public`, which must be specified in the subclass that implements them (subclasses must override and implement non-default interface methods).

Interfaces cannot be instantiated. (ie. `Friendly f = new Friendly();` does not compile)

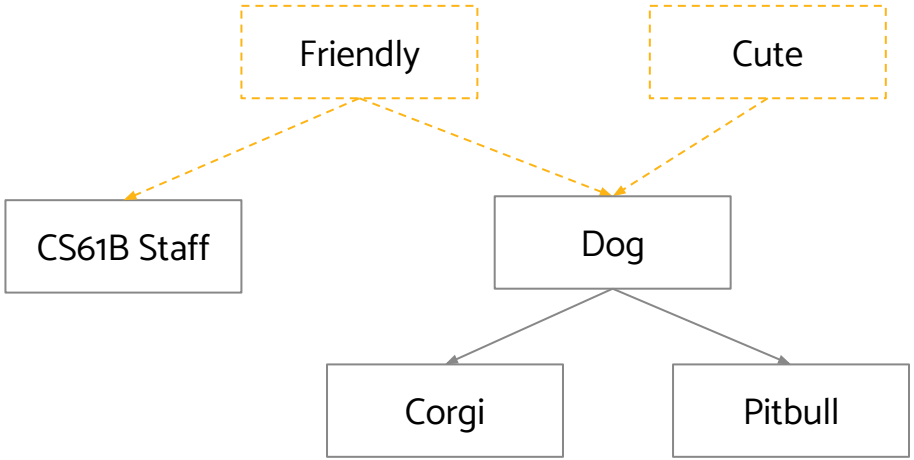


Interfaces vs. Classes

- A class can **`implement`** many interfaces and **`extend`** only one class
- Interfaces tell us what we want to do but not how; classes tell us how we want to do it
- Interfaces can have empty method bodies (that must be filled in by subclasses) or default methods (do not need to be overridden by subclasses)
- With extends, subclasses inherit their parent's instance and static variables, methods (can be overridden), nested classes
    - But not constructors!
    - Use **`super`** to refer to the parent class

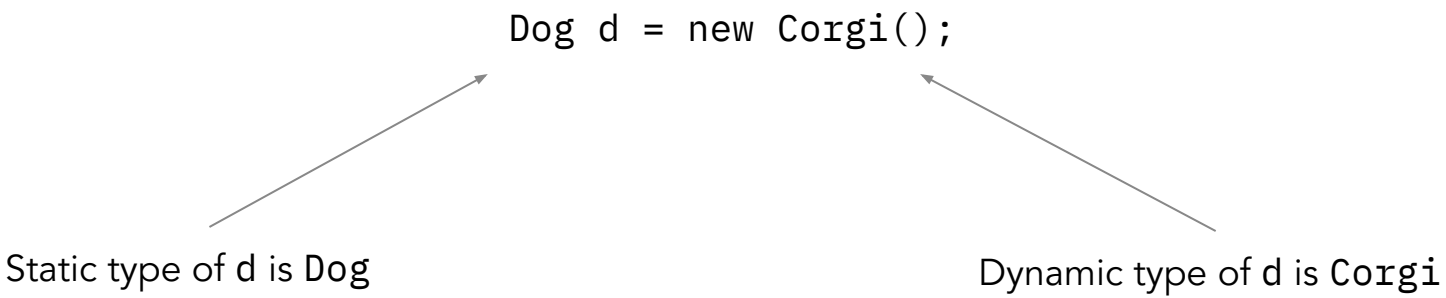
Implementation

```
interface Cute {...}

interface Friendly {...}

class CS61BStaff implements Friendly {...}

class Dog implements Cute, Friendly {...}

class Corgi extends Dog {...}

class Pitbull extends Dog {...}
```



Static vs. Dynamic Type

A variable's static type is specified at declaration, whereas its dynamic type is specified at instantiation (e.g. when using new).



The static and dynamic type of a variable have to complement each other or else the code will error. For example, a `Dog` is not necessarily a `Corgi`, so `Corgi c = new Dog();` will not compile.

General rule of thumb: Given LHS = RHS, is RHS guaranteed to be a LHS?

Though interfaces cannot be instantiated, they can be static types (ie. `Cute c = new Corgi();`)

Casting

Casting allows us to tell the compiler to treat the static type of some variable as whatever we want it to be (need to have a superclass/subclass relationship). If the cast is valid, for that line only we will treat the static type of the casted variable to be whatever we casted it to.

```
Animal a = new Dog();
Dog d = a;     // Compiler error: an animal is not a dog
Dog d = (Dog) a;     // Valid cast: an animal could reasonably be a dog
d = new Dog();
a = (Animal) d;    // Valid cast: a dog definitely is an animal
Cat c = new Cat();
d = (Dog) c;      // Compiler error: a cat is definitely not a dog
a = c;
d = (Dog) a;    // Cast compiles because an animal could reasonably be a dog.
                   During runtime, errors
```

All these concepts - What's the point?

It allows for Subtype Polymorphism. (You'll also see this in lecture this week).
Polymorphism means "providing a single interface to entities of different types"

Example:
Consider a variable deque of static type `Deque`:
When you call `deque.addFirst()`, the actual behavior is based on the dynamic type.

```
Deque deque = new LinkedListDeque();// Runs LinkedListDeque's addFirst
Deque deque = new ArrayDeque();// Runs ArrayDeque's addFirst
```

Java automatically selects the right behavior using what is sometimes called "dynamic method selection".

Dynamic Method Selection

Your computer. . .

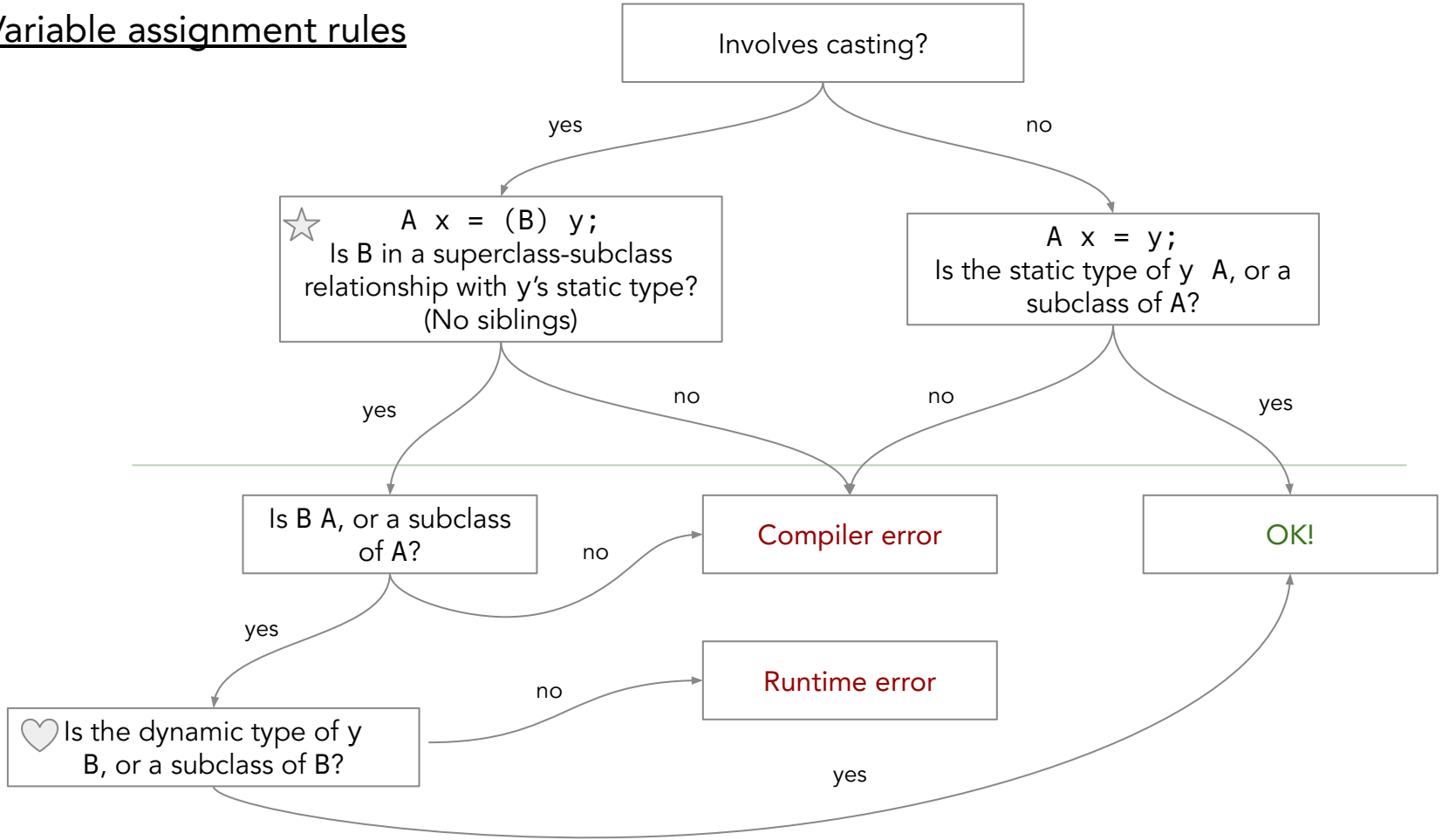
@ Compile Time, we only care about static type of the invoking / calling instance:

1. Check for valid variable assignments
2. Check for valid method calls (only considering static type and static types superclass(es))
    a. Lock in exact method signature as soon as we find an adequate one, traversing parent classes
3. If nothing found, compiler error

@ Run Time, we care about dynamic type of the invoking / calling instance:

1. If the locked-in method is static, skip the step below and just run that method
2. Ensure casted objects can be assigned to their variables
3. Check for overridden methods
    a. Start with dynamic type, if found overridden method run it. Otherwise traverse up to parent classes.

Variable assignment rules



Worksheet

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

```
public class CatBus ____________________ __________, __________ {
      @Override
      __________ __________ __________ {
            // CatBus revs its engine, implementation not shown
      }

      @Override
      __________ __________ __________ {
            // CatBus honks, implementation not shown
      }

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

Fill in the CatBus class so CatBuses can rev their engines and honk at other CatBuses.

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

```
public class CatBus implements Honker, Vehicle {
      @Override
      ---------- ---------- ---------- {
            // CatBus revs its engine, implementation not shown
      }

      @Override
      ---------- ---------- ---------- {
            // CatBus honks, implementation not shown
      }

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

Fill in the CatBus class so CatBuses can rev their engines and honk at other CatBuses.

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

```
public class CatBus implements Honker, Vehicle {
      @Override
      public void revEngine() {
            // CatBus revs its engine, implementation not shown
      }

      @Override
      ---------- ---------- ---------- {
            // CatBus honks, implementation not shown
      }

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

Fill in the CatBus class so CatBuses can rev their engines and honk at other CatBuses.

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

```
public class CatBus implements Honker, Vehicle {
      @Override
      public void revEngine() {
            // CatBus revs its engine, implementation not shown
      }

      @Override
      public void honk() {
            // CatBus honks, implementation not shown
      }

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

Fill in the CatBus class so CatBuses can rev their engines and honk at other CatBuses.

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

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

Update the `conversation` method signature so that CatBuses can honk at CatBuses *and* Gooses while only having one argument, `target`.

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

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

Update the `conversation` method signature so that CatBuses can honk at CatBuses *and* Gooses while only having one argument, `target`.

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

Assume that `CatBus` and `Goose` use the default constructor. Which of the following lines will compile?

```
Honker cb = new CatBus();
CatBus g = new Goose();
Honker h = new Honker();
CanadaGoose cg = new Goose();
Honker hcg = new CanadaGoose();
```

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

Assume that `CatBus` and `Goose` use the default constructor. Which of the following lines will compile?

```
Honker cb = new CatBus(); // Compiles - a CatBus is a kind of Honker
CatBus g = new Goose();
Honker h = new Honker();
CanadaGoose cg = new Goose();
Honker hcg = new CanadaGoose();
```

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

Assume that `CatBus` and `Goose` use the default constructor. Which of the following lines will compile?

```
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();
CanadaGoose cg = new Goose();
Honker hcg = new CanadaGoose();
```

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

Assume that `CatBus` and `Goose` use the default constructor. Which of the following lines will compile?

```
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();
Honker hcg = new CanadaGoose();
```

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

Assume that `CatBus` and `Goose` use the default constructor. Which of the following lines will compile?

```
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, not
                                     necessarily the other way around
Honker hcg = new CanadaGoose();
```

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

Assume that `CatBus` and `Goose` use the default constructor. Which of the following lines will compile?

```
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, not
                                     necessarily the other way around
Honker hcg = new CanadaGoose(); // Compiles - a CanadaGoose is a kind of Honker
```

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`			
`a.greet(c);`			
`a.sleep();`			
`c.play()`			
`c.greet(d);`			
`((Animal) c).greet(d);`			
`d.sleep();`			
`a = c;`			
`a.play(14);`			
`((Cat) b).play();`			
`d = (Dog) a;`			
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE

```
a.greet(c);
a.sleep();
c.play()
c.greet(d);
((Animal) c).greet(d);
d.sleep();
a = c;
a.play(14);
((Cat) b).play();
d = (Dog) a;
c = a;
```

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"

```
a.sleep();
c.play()
c.greet(d);
((Animal) c).greet(d);
d.sleep();
a = c;
a.play(14);
((Cat) b).play();
d = (Dog) a;
c = a;
```

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`			
`c.greet(d);`			
`((Animal) c).greet(d);`			
`d.sleep();`			
`a = c;`			
`a.play(14);`			
`((Cat) b).play();`			
`d = (Dog) a;`			
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play();`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`			
`((Animal) c).greet(d);`			
`d.sleep();`			
`a = c;`			
`a.play(14);`			
`((Cat) b).play();`			
`d = (Dog) a;`			
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play();`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"

```
((Animal) c).greet(d);
d.sleep();
a = c;
a.play(14);
((Cat) b).play();
d = (Dog) a;
c = a;
```

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`((Animal) c).greet(d);`	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`d.sleep();`			
`a = c;`			
`a.play(14);`			
`((Cat) b).play();`			
`d = (Dog) a;`			
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`((Animal) c).greet(d);`	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`d.sleep();`	Dog's sleep()	N/A, sleep() is static	"I love napping!"

```
a = c;
a.play(14);
((Cat) b).play();
d = (Dog) a;
c = a;
```

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`((Animal) c).greet(d);`	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`d.sleep();`	Dog's sleep()	N/A, sleep() is static	"I love napping!"
`a = c;`	ok	ok	[no output]
`a.play(14);`			
`((Cat) b).play();`			
`d = (Dog) a;`			
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`((Animal) c).greet(d);`	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`d.sleep();`	Dog's sleep()	N/A, sleep() is static	"I love napping!"
`a = c;`	ok	ok	[no output]
`a.play(14);`	Error	N/A	Compiler error
`((Cat) b).play();`			
`d = (Dog) a;`			
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`((Animal) c).greet(d);`	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`d.sleep();`	Dog's sleep()	N/A, sleep() is static	"I love napping!"
`a = c;`	ok	ok	[no output]
`a.play(14);`	Error	N/A	Compiler error
`((Cat) b).play();`	Cat's play()	Error	Runtime error
`d = (Dog) a;`			
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`((Animal) c).greet(d);`	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`d.sleep();`	Dog's sleep()	N/A, sleep() is static	"I love napping!"
`a = c;`	ok	ok	[no output]
`a.play(14);`	Error	N/A	Compiler error
`((Cat) b).play();`	Cat's play()	Error	Runtime error
`d = (Dog) a;`	ok	Error	Runtime error
`c = a;`			

2A Raining Cats and Dogs

```
Animal a = new Dog("Pluto");
Animal b = new Animal("Bear");
Cat c = new Cat("Garfield");
Dog d = new Dog("Lucky");
```

	Compile Time (static)	Runtime (dynamic)	Output
`Cat e = new Animal("Kitty");`	Error	N/A	CE
`a.greet(c);`	Animal's greet(Animal)	Dog's greet(Animal)	"Dog Pluto says: Woof!"
`a.sleep();`	Animal's sleep()	N/A, sleep() is static	"Naptime!"
`c.play()`	Cat's play()	Cat's play()	"Woo it is so much fun being a cat! Meow!"
`c.greet(d);`	Cat's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`((Animal) c).greet(d);`	Animal's greet(Animal)	Cat's greet(Animal)	"Cat Garfield says: Meow!"
`d.sleep();`	Dog's sleep()	N/A, sleep() is static	"I love napping!"
`a = c;`	ok	ok	[no output]
`a.play(14);`	Error	N/A	Compiler error
`((Cat) b).play();`	Cat's play()	Error	Runtime error
`d = (Dog) a;`	ok	Error	Runtime error
`c = a;`	Error	N/A	Compiler error

2B Raining Cats and Dogs

How might we fix the error in the line assigning `c = a`?

2B Raining Cats and Dogs

How might we fix the error in the line assigning `c = a`?

- We could fix this error by casting a to be a Cat: `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.