# CS61B Discussion 4

Inheritance, Static and Dynamic Typing

# CS61B Discussion 4

## Announcements

- Project 0 due Fri 2/13 at 11:59pm
  24 slip hours (unused hours roll over)
- HW3 released tonight, due Tues 2/17 at 10:00pm
- Midterm 1 on Wed 2/18 at 6:00pm
- Interview prep resources will come as soon as I have time to compile them (target: after Midterm 1)

# Survey Outtakes

**Areas I Will Work On:**

- Writing bigger (remind me!)
- Improve pacing – this may mean less individual work time
- Will cold call if no volunteers (so work with each other)

**Important:**

- If you're feeling behind, please reach out! Email, office hours, individual appointments.

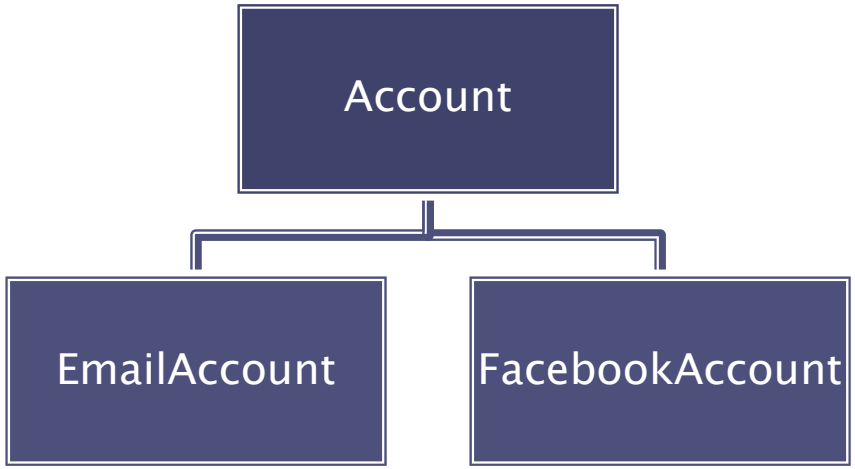
# Access Modifiers

| Modifier | Own Class | Package | Subclass | Everywhere |
|---|---|---|---|---|
| `public` | ✓ | ✓ | ✓ | ✓ |
| `protected` | ✓ | ✓ | ✓ | ✗ |
| default (package) | ✓ | ✓ | ✗ | ✗ |
| `private` | ✓ | ✗ | ✗ | ✗ |

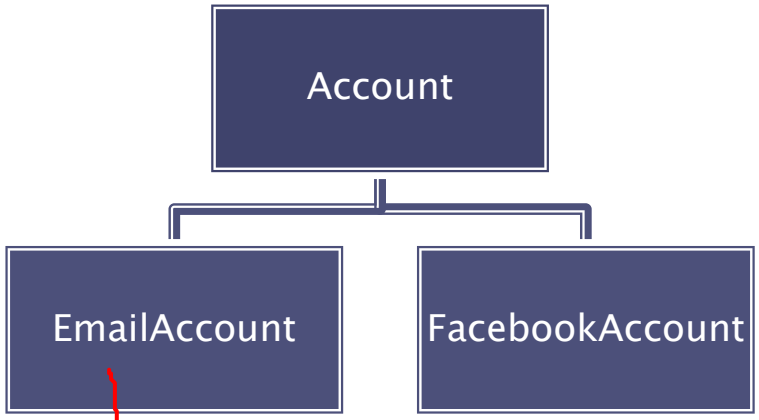
# Inheritance Basics

- Models IS-A relationship between classes
- Subclass can override behavior in superclass

  Match the method signature exactly.
- Dynamic Method Lookup:
  - Search for method in own class (dynamic type).
  - Search for method in parent class (of dynamic type).
  - All objects in Java inherit from `Object`.



# Inheritance Basics




```java
public class Account {
  protected String username, password;
  public Account(String username, String password)
    this.username = username; this.password = password;
  }
  public boolean login(String password) { /** CODE HERE **/ }
}

public class EmailAccount extends Account {
  public EmailAccount(String username, String password) {
    super(username, password);
  }
  public void fetchEmail() { /** CODE HERE **/ }
}

public class FacebookAccount extends Account {
  public FacebookAccount(String username, String password) {
    super(username, password);
    loadNewsFeed();
  }
  public void loadNewsFeed() { /** CODE HERE **/ }
  public void pokeFriend(Friend f) { /** CODE HERE **/ }
}
```

# Q1: Creating Cats

```java
public class Animal {
    protected String name, noise;
    protected int age;

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

    public String makeNoise() {
        if (age < 5) {
            return noise.toUpperCase();
        } else {
            return noise;
        }
    }

    public void greet() {
        System.out.println("Animal " + name + " says: " + makeNoise());
    }
}
```

# Q1: Creating Cats

## Observations

- Cat IS-A Animal (so extend)
- Cat has different noise – so override value
- Cat has different behavior for greet() – so override method
- All other behavior is the same

| Animal | |
|---|---|
| String<br>String<br>int | name<br>noise<br>age |
| String<br>void | makeNoise()<br>greet() |

| Cat | |
|---|---|
| void | greet() |

# Q1: Creating Cats (Solution)

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

    @Override
    public void greet() {
        System.out.println("Cat " + name + " says: " + makeNoise());
    }
}
```

What if we had this as our constructor?

```
public Cat(String name, int age) {
  this.noise = "Meow!";
  super(name, age);
}
```

What does this annotation do?

Where is makeNoise() defined?

## So what does this print?

```
public class Tester {
  public static void main(String[] args) {
    Animal a = new Animal("Fido", 5);
    a.greet();
    Cat c = new Cat("Garfield", 10);
    c.greet();
  }
}
```

# Static vs Dynamic Types

- **Static Type**: Declared type of variable; known at compile time.

  Used by compiler to type-check.

- **Dynamic Type**: Actual type of variable; known only at run time

  Used at runtime to call methods.

- **Cast** changes static type and is *promise to compiler* that you will provide compatible type. Still executes based on dynamic type at runtime.

# Static vs Dynamic Types

- **Example 1**: Assign specific to more general variable.

```
Animal a = new Cat("Garfield", 10);
a.greet();
```

✓ Compiles and runs correctly. Calls Cat's greet() method.

- **Example 2**: Assign general to more specific variable.

```
Cat c = new Animal("Pikachu", 10);
```

× Compiler error.

- **Example 3**: Casting

```
Animal a = new Cat("Garfield", 10);
Cat c = (Cat) a;
```

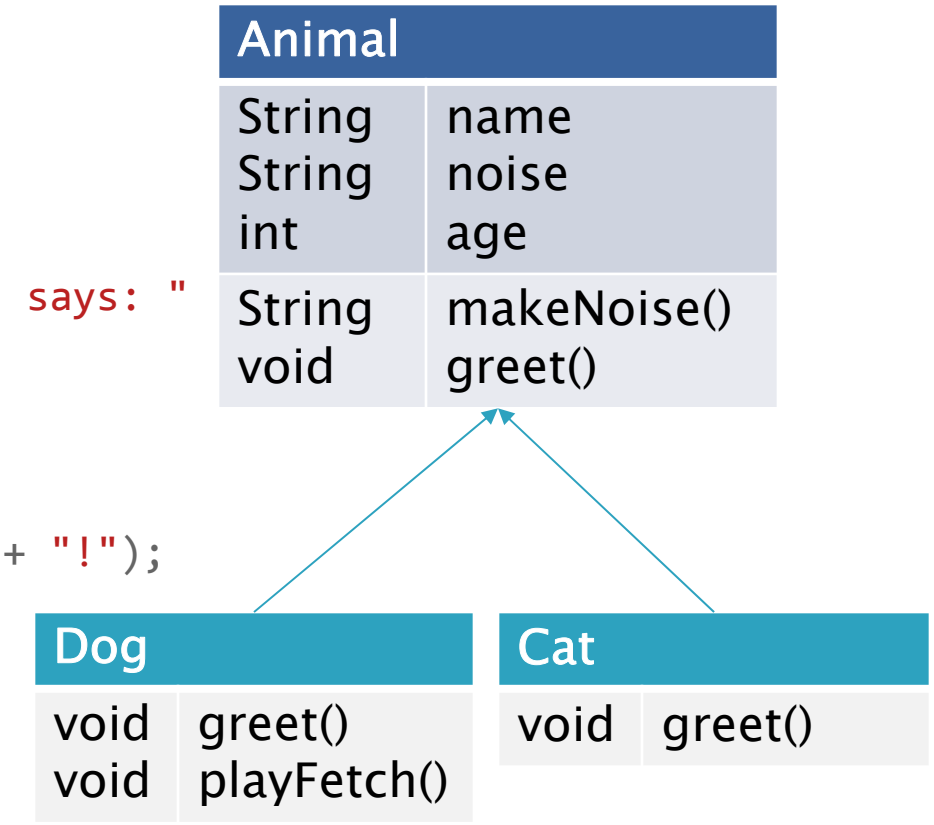
✓ Compiles correctly; potential runtime error.

# Q2: Raining Cats and Dogs

```java
public class Dog extends Animal {
  public Dog(String name, int age) {
    super(name, age);
    noise = "Woof!";
  }

  @Override
  public void greet() {
    System.out.println("Dog " + name + " says: "
        + makeNoise());
  }

  public void playFetch() {
    System.out.println("Fetch, " + name + "!");
  }
}
```



# Q2: Raining Cats and Dogs

```
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();            // (A) _____________________
    c.greet();            // (B) _____________________
    d.greet();            // (C) _____________________

    a = c;
    a.greet();            // (D) _____________________
    ((Cat) a).greet();    // (E) _____________________
  }
}
```

| | Static Type | Dynamic Type |
|---|---|---|
| a | Animal | |
| c | Cat | |
| d | Dog | |

# Q2: Raining Cats and Dogs

```
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();            // (A) Animal Pluto says: Huh?
    c.greet();            // (B) ______________________
    d.greet();            // (C) ______________________

    a = c;
    a.greet();            // (D) ______________________
    ((Cat) a).greet();    // (E) ______________________
  }
}
```

| | Static Type | Dynamic Type |
|---|---|---|
| a | Animal | Animal |
| c | Cat | |
| d | Dog | |

# Q2: Raining Cats and Dogs

```
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();            // (A) Animal Pluto says: Huh?
    c.greet();            // (B) Cat Garfield says: Meow!
    d.greet();            // (C) ______________________

    a = c;
    a.greet();            // (D) ______________________
    ((Cat) a).greet();    // (E) ______________________
  }
}
```

| | Static Type | Dynamic Type |
|---|---|---|
| a | Animal | Animal |
| c | Cat | Cat |
| d | Dog | |

# Q2: Raining Cats and Dogs

```
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();            // (A) Animal Pluto says: Huh?
    c.greet();            // (B) Cat Garfield says: Meow!
    d.greet();            // (C) Dog Fido says: WOOF!

    a = c;
    a.greet();            // (D) ____________________
    ((Cat) a).greet();    // (E) ____________________
  }
}
```

| | Static Type | Dynamic Type |
|---|---|---|
| a | Animal | Animal |
| c | Cat | Cat |
| d | Dog | Dog |

# Q2: Raining Cats and Dogs

```
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();          // (A) Animal Pluto says: Huh?
    c.greet();          // (B) Cat Garfield says: Meow!
    d.greet();          // (C) Dog Fido says: WOOF!

    a = c;
    a.greet();          // (D) Cat Garfield says: Meow!
    ((Cat) a).greet();  // (E) ____________________
  }
}
```

| | Static Type | Dynamic Type |
|---|---|---|
| a | Animal | ~~Animal~~ Cat |
| c | Cat | Cat |
| d | Dog | Dog |

# Q2: Raining Cats and Dogs

```java
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();          // (A) Animal Pluto says: Huh?
    c.greet();          // (B) Cat Garfield says: Meow!
    d.greet();          // (C) Dog Fido says: WOOF!

    a = c;
    a.greet();          // (D) Cat Garfield says: Meow!
    ((Cat) a).greet();  // (E) Cat Garfield says: Meow!
  }
}
```

| | Static Type | Dynamic Type |
|---|---|---|
| a | Animal | Cat |
| c | Cat | Cat |
| d | Dog | Dog |

a = d;

((Dog) a).playFetch();

# Q2: Raining Cats and Dogs

```
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();          // (A) Animal Pluto says: Huh?
    c.greet();          // (B) Cat Garfield says: Meow!
    d.greet();          // (C) Dog Fido says: WOOF!

    a = c;
    a.greet();          // (D) Cat Garfield says: Meow!
    ((Cat) a).greet();  // (E) Cat Garfield says: Meow!

    a = new Dog("Hieronymus", 10);
    d = a;
  }
}
```

# Q2: Raining Cats and Dogs

```
public class TestAnimals {
  public static void main(String[] args) {
    Animal a = new Animal("Pluto", 10);
    Cat c = new Cat("Garfield", 6);
    Dog d = new Dog("Fido", 4);

    a.greet();            // (A) Animal Pluto says: Huh?
    c.greet();            // (B) Cat Garfield says: Meow!
    d.greet();            // (C) Dog Fido says: WOOF!

    a = c;
    a.greet();            // (D) Cat Garfield says: Meow!
    ((Cat) a).greet();    // (E) Cat Garfield says: Meow!

    a = new Dog("Hieronymus", 10);
    ~~d = a;~~ d = (Dog) a;
  }
}
```

Will this work at runtime?

# Behavioral Summary for Inheritance

**Invocation of overridden methods:**

- Compiler plays it safe and only lets us do things allowed by ***static*** type.
- The actual method invoked is based on **dynamic** type.

**Invocation of hidden static methods or hidden variables:**

- Actual method invoked or variable accessed is based on ***static*** type.

**Melanie's awesome cheatsheet:** Piazza @1476

Source: CS61B Sp'15 Lecture 9 Slide 35

# Static Field Lookup, Dynamic Method Lookup

- See Melanie's cheatsheet: Piazza @1476
- Calling an **instance method**:
  - Check: Static type (or any parent) has method signature. (Else: Compile-Time Error)
  - Lookup: Start from dynamic type. If method found, use it, else search parent. (Repeat)
- Calling a **static method**:
  - Check: Same as above.
  - Lookup: Start from static type, or use parent.
- Invocation of a **hidden variable**:
  - Access is based on static type of variable or method being called.

Sources: Melanie's Cheatsheet (@1476), CS61B Sp'15 Lecture 9 Slide 35