CISC 3120

C10: Garbage Collection and Constructors

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Recap
  - OOP in Java: composition & inheritance
  - Project 1 and lessons learned?
- Memory management
- Java garbage collection
- Constructors
- Assignments

More about Stack and Heap

- Two important region of memories
  - The Stack
  - The Heap

Stack

- Methods are "stacked"
- Stack is organized as stack frames
  - A stack frame holds the state of the method (method invocation and automatic data)
    - Program counter: which line of code being executed
    - Automatic data: values of method parameters and local variables

Stack: Example

- doStuff gets called …

1

doStuff()	b

2

go(4)	x, z
doStuff()	b

3

crazy()	c
go(4)	x, z
doStuff()	b

4

go(4)	x, z
doStuff()	b

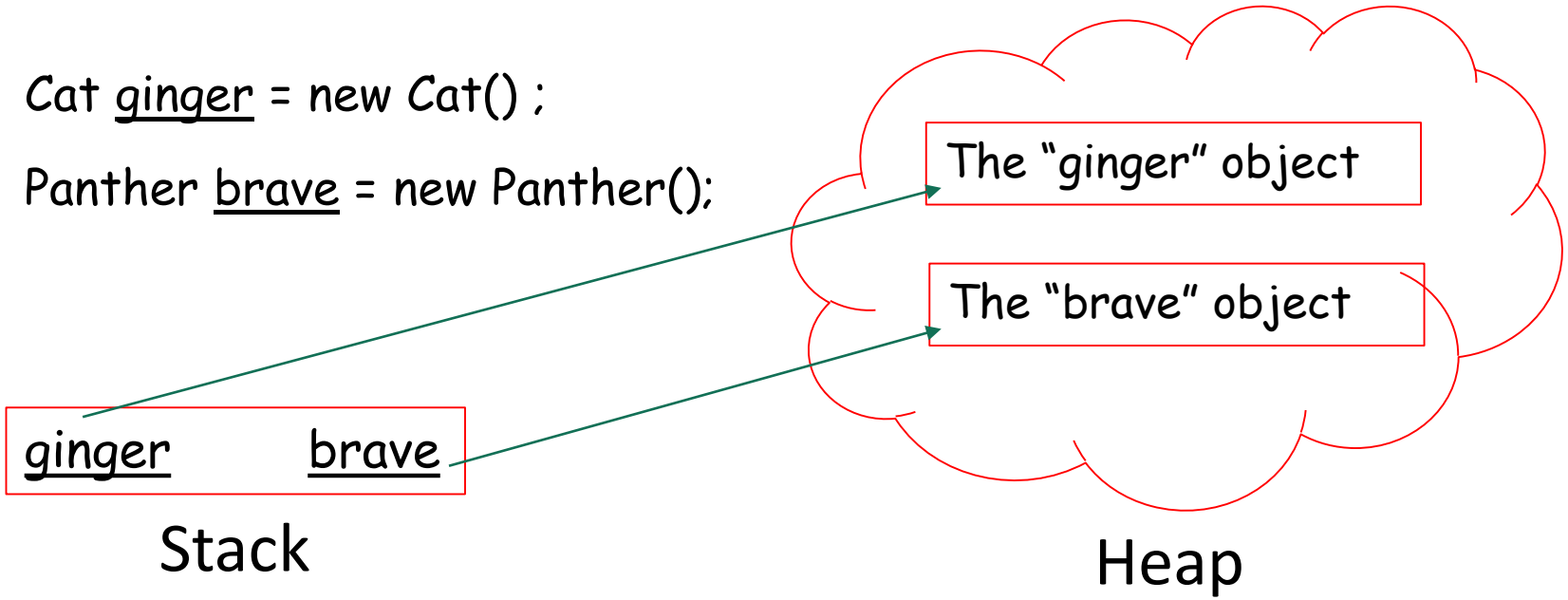
```
void doStuff() {
    boolean b = true;
    go(4);
}
void go(int x) {
    int z = x + 24;
    crazy();
}
Void crazy() {
   int c = 36;
}
```

Heap

- Objects including their instance variables live

Cat ginger = new Cat() ;

Panther brave = new Panther();



Questions?

- Stack and Heap

Program Data

- We restrict the definition of program data to data associated with variables
  - Data are stored in the memory, and "referenced" via variables
  - Variables are names assigned to allocated memory
  - Program data: memory allocations of variables or referenced by the variables
- In C++ and Java, program data are in three categories
  - Automatic
  - Static
  - Dynamic

Automatic, Static, and Dynamic Program Data

- They differ in
  - (where) which region of memory the data reside
  - (when) when and how the data is allocated in memory
  - (when) when and how the data is deallocated in memory
- Variables
  - where: scope: where it can be accessed, related to where it is allocated
  - when: lifetime: when it is allocated and when it is deallocated

Automatic Data

- Memory is automatically allocated and deallocated for automatic data
- Where: the memory is allocated in a region of memory called the *stack*
- When to allocate: the memory is allocated when execution reaches the scope of the variable
- When to deallocate: the memory is deallocated when execution leaves the scope of the variable

Automatic Data: Examples

- In C++ and Java: where are the variables for automatic data?

```
int sumToNumber(int number) {
    int sum = 0;
    for (int i=0; i<=number; i++) {
        sum += i;
    }
    return sum;
}
```

Automatic Data: Examples

- In C++ and Java: where are the variables for automatic data?
  - Parameter: number
  - Local variables:
    - sum
    - i
  - What are their scopes?

```
int sumToNumber(int number) {
    int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
    return sum;
}
```

Automatic Data: Examples

- In C++ and Java: where are the variables for automatic data?
  - What are their scopes?

```
int sumToNumber(int number) {
    int sum = 0;
    for (int i=0; i<=number; i++) {
        sum += i;
    }
    return sum;
}
```

i

sum

number

Automatic Data: C++ and Java

- C++ permits automatic object allocation while Java does not
- Example: the effect in C++ and that in Java are different

```
void method() {

   Cat cat;

}
```

  - In C++, a Cat object is allocated and instantiated in the Stack; in Java, only a reference variable in the Stack

Automatic Data: C++ and Java

- In C++, object is allocated and instantiated in the Stack; in Java, only a reference variable in the Stack

C++

```
void method() {
   Cat cat;
}
```

$\neq$

Java

```
void method() {
   Cat cat;
}
```

```
void method() {
   Cat *cat;
}
```

$\approx$

```
void method() {
   Cat cat;
}
```

Questions?

- Memory, program data, and variables
- Automatic program data

Static Data

- Static data's existence does not change during the entire execution of a program
- Where:
  - the memory for static data is allocated in a region of memory, generically referred to as the static data segment
- When to allocate:
  - the memory is allocated when the program starts,
  - or when execution reaches the static variable declaration for the first time
- When to deallocate:
  - the memory for static data is deallocated when the program exits

Static Data: Example in C++

- In C++: where are the variables for static data?

```
int sumToNumber(int number) {
    static int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
    return sum;
}
```

Static Data: Example in C++

- In C++: where are the variables for static data?
  - Local variable: sum
  - What is its scope?

```
int sumToNumber(int number) {
    static int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
    return sum;
}
```

Static Data: Example in C++

- In C++: where are the variables for static data?
  - What are their scopes?

```
int sumToNumber(int number) {
    static int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
}
```

sum

Static and Automatic Data: Example in C++

- In C++: when are they allocated and deallocated?

```
int sumToNumber(int number) {
    static int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
    return sum;
}
```

```
int sumToNumber(int number) {
    int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
    return sum;
}
```

Static and Automatic Data: Example in C++

- How do the lifetimes of the automatic and static variables differ?
- Compare them in running programs
  - When sum is static
  - When sum is automatic

```
cout << sumToNumber(5) << endl;
cout << sumToNumber(5) << endl;
```

Static Data in C++

- In C++
  - Variables declared as “static”
  - Additionally, variables declared outside any function and class body

Static Data: Example in Java

- Can we write the following in Java?

```
int sumToNumber(int number) {
    static int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
    return sum;
}
```

Static Data in Java

- Can we write the following in Java?

```
int sumToNumber(int number) {
    static int sum = 0;
    for (int i=0; i<number; i++) {
        sum += i;
    }
    return sum;
}
```

Static Data in Java

- Java is more restrictive
  - Static variables can only declared within a class, but not within any methods
- Static variables are class variables with the scope of the class
- There exists exactly one incarnation of the field, no matter how many instances (possibly zero) of the class may eventually be created

Static Data: Example in Java

- Where are the static variables?

```
class StaticSum {
    static int sum = 0;
    int sumToNumber(int number) {
        for (int i=0; i<number; i++) {
            sum += i;
        }
        return sum;
    }
}
```

Static Data: Example in Java

- What is its scope?

```
class StaticSum {
    static int sum = 0;
    int sumToNumber(int number) {
        for (int i=0; i<number; i++) {
            sum += i;
        }
        return sum;
    }
}
```

Static Data: Example in Java

- What is the output of this program?

```
class StaticSum {
        public static void main(String[] args) { StaticSum s = new StaticSum();
             System.out.println(s.sumToNumber(5)); System.out.println(s.sumToNumber(5));
        }
        int sumToNumber(int number) {
            for (int i=0; i<number; i++) {
                sum += i;
            }
            return sum;
        }
        static int sum = 0;
}
```

Static Data in Java

- Java is more restrictive
  - Static variables can only declared within a class, but not within any methods
- Static variables are class variables with the scope of the class
- Static variables has globe scope
- Static variables has the lifetime of the program
- Local and inner classes are only allowed with final static variables
  - To ensure that there exists exactly one incarnation of the field, no matter how many instances (possibly zero) of the class may eventually be created

Questions?

- Static program data
- Difference between C++ and Java in terms of static program data

Dynamic Data

- Programmers are responsible for allocating dynamic data
- In C++: programmers are also responsible for deallocating the dynamic data.
- Where: the memory for static data is allocated in a region of memory, generically referred to as the *heap*
- When to allocate:
  - the memory is allocated when the programmer invokes the "new" operator.
- When to deallocate:
  - In Java: when the Java Garbage Collector reclaims the object allocated
  - In C++: when the programmer invokes the delete operator to free the memory allocated to the dynamic data

Dynamic Data in C++ and Java

- In C++, programmers can allocate memory for any data types, i.e., to use "new" operator against any data types
- In Java, programmers can only use "new" operator for reference data types

Dynamic Data in C++ and Java: Examples

- In Java: which one of the following is legal?

```
int  i = new int;
```

```
Cat ginger = new Cat();
```

```
int[] iArr = new int[10];
```

- In C++: which one of the following is legal?

```
int *iPtr  = new int;
```

```
Cat *gingerPtr = new Cat();
```

```
int* iArr = new int[10];
```

Dynamic Data in C++ and Java: Examples

- In Java: which one of the following is legal?

```
int  i = new int;
```

```
Cat ginger = new Cat();
```

```
int[] iArr = new int[10];
```

- In C++: which one of the following is legal?

```
int *iPtr  = new int;
```

```
Cat *gingerPtr = new Cat();
```

```
int* iArr = new int[10];
```

Dynamic Data: Java and C++ Comparison

- Allocation:
  - The same
    - Java and C++: dynamic data are created using the new operator
  - The different
    - In Java: dynamic data can only be objects, cannot be primitive data types.
    - In C++: dynamic data can be any data types
- Deallocation
  - The different
    - In C++: programmers use the delete operator to deallocate memory
    - In Java: Java Garbage Collector is responsible for deallocating the memory, programmers have little control.

Objects in Java and C++

- C++ can have both automatically and dynamically allocated objects

```
Cat *ginger = new Cat();
ginger->pounce();
(*ginger).pounce();
```

```
Cat ginger;
ginger.pounce();
```

- Java has only dynamically allocated objects

```
Cat ginger;
ginger.pounce();
```

```
Cat ginger = new Cat()
ginger.pounce();
```

Objects in Java and C++

- C++ can have both automatically and dynamically allocated objects

```
Cat ginger;

ginger.pounce();
```

```
Cat *ginger = new Cat();

ginger->pounce();

(*ginger).pounce();
```

- Java has only dynamically allocated objects

```
Cat ginger;

ginger.pounce();
```

```
Cat ginger = new Cat()

ginger.pounce();
```

Dynamic Data: Programming Error in C++

- Programmers are responsible for managing dynamic data, which is error-prone
- Common errors
  - Inaccessible objects
  - Memory leaks
  - Dangling pointers
- A little bit more about C++ memory management next

Memory Management in C++

- In C++: compare the following two.

```
Cat *ginger = new Cat();

ginger = nullptr;
```

```
Cat *ginger = new Cat();

delete ginger;

ginger = nullptr;
```

Memory Management in C++

- In C++: compare the following two.

```
Cat *ginger = new Cat();

ginger = nullptr;
```

```
Cat *ginger = new Cat();

delete ginger;

ginger = nullptr;
```

- A programmer must <u>explicitly</u> free the memory of an object allocated in the Heap in a C++ program; otherwise, the memory <u>cannot</u> be reclaimed and used in the program.

Complexity of Memory Management in C++

- Automatic and Heap storage in C++

```
Cat ginger("ginger");
Cat tuxedo("tuxedo");
ginger.tap(tuxedo);
// programmers don't free memory
```

```
Cat *ginger = new Cat("ginger");
Cat tuxedo;
ginger->tap(tuxedo);
delete ginger; // must free memory
```

- How about the following example?

```
Cat ginger;
Cat *catptr = &ginger;
delete catptr;
```

Complexity of Memory Management in C++

- Automatic and Heap storage in C++

```
Cat ginger("ginger");
Cat tuxedo("tuxedo");
ginger.tap(tuxedo);
// programmers don't free memory
```

```
Cat *ginger = new Cat("ginger");
Cat tuxedo;
ginger->tap(tuxedo);
delete ginger; // must free memory
```

- How about the following example?

```
Cat ginger;
Cat *catptr = &ginger;
delete catptr;
```

Memory Management in Java

- In Java:

```
Cat ginger = new Cat();
Cat tuxedo = new Cat();
ginger.tap(tuxedo);
```

```
Cat ginger = new Cat();
ginger.tap(new Cat());
```

  - Programmers generally do not deal with reclaiming the memory.
  - Java Garbage Collector takes care of it.

Memory Management in Java

- In Java:

```
Cat ginger = new Cat();
Cat tuxedo = new Cat();
ginger.tap(tuxedo);
```

```
Cat ginger = new Cat();
ginger.tap(new Cat());
```

- In C++

```
Cat *ginger = new Cat();
Cat *tuxedo = new Cat();
ginger->tap(*tuxedo);
delete ginger;
delete tuxedo;
```

```
Cat *ginger = new Cat();
ginger->tap(*(new Cat()));
delete ginger
//  how about the other cat?
```

Memory Management in Java

- In Java:

```
Cat ginger = new Cat();
Cat tuxedo = new Cat();
ginger.tap(tuxedo);
```

```
Cat ginger = new Cat();
ginger.tap(new Cat());
```

- In C++

```
Cat *ginger = new Cat();
Cat *tuxedo = new Cat();
ginger->tap(*tuxedo);
delete ginger;
delete tuxedo;
```

```
Cat *ginger = new Cat();
ginger->tap(*(new Cat()));
delete ginger
//  how about the other cat?
```

Questions?

- Dynamic data in Java and C++
- Review C++ and Java memory management
- How they affect the ways to write programs
  - JVM takes away some responsibility from programmers
- Next, we shall discuss more about program data, memory management/JVM, and constructor

Java Garbage Collector

- Java is responsible for deallocating dynamic data, and programmers are not.
- In Java, we often write

```
Cat ginger = new Cat();

ginger.pounce(new Animal());
```

- In C++, we never write (although it compiles)

```
Cat *ginger = new Cat();

ginger->pounce(new Animal());
```

Different Garbage Collection Algorithms

- How does a Garbage Collector figure out an object is no longer needed and can be deallocated?
- Reference counting
- Trace-based garbage collector
  - e.g., Baker's algorithm
  - Copying collector

Advantage of Garbage Collection

- Avoid bugs, such as,
  - Forget to free memory (memory leak)
  - Use already freed objects (dangling pointers)
  - Also in Java, programmers do not have direct memory access, and cannot accidentally overwrite memory.

Disadvantage of Garbage Collection

- Consume resources (memory and processor)
- Unpredictable stalls
- Memory leak still possible, but harder to understand
- No manual control

Questions

- Concept of Garbage Collector
- Programming in Java that does garbage collection

Constructors

- Like C++, constructors in Java
  - have the identical name as the name of the class,
  - do not specify return type,
  - are called when an object is created,
  - and are responsible for initializing the object (instance variables)

Default Constructor

- Java compiler provides the default constructor when no constructor is written.

```
class Cat {
    void pounce(Cat otherCat) {…}
}
```

```
Cat ginger = new Cat(); // calling default constructor
ginger.pounce(new Cat());
```

Default Constructor?

```
class Cat {
    private String name;
    public Cat(String name) {this.name = name;}
    void pounce(Cat otherCat) { … }
}
```

- Can we use the default constructor now?

```
Cat ginger = new Cat();
ginger.pounce(new Cat("tiger"));
```

Default Constructor?

```
class Cat {
    private String name;
    public Cat(String name) {this.name = name;}
    void pounce(Cat otherCat) { … }
}
```

- Can we use the default constructor now?

```
Cat ginger = new Cat();
ginger.pounce(new Cat("tiger"));
```

Default and Parameterized Constructors

- Java ceases to create the default constructor when a constructor is provided

```
class Cat {
    private String name;
    public Cat() {name = "cat";}
    public Cat(String name) {this.name = name;}
    void pounce(Cat otherCat) { … }
}
```

```
Cat ginger = new Cat(); …
```

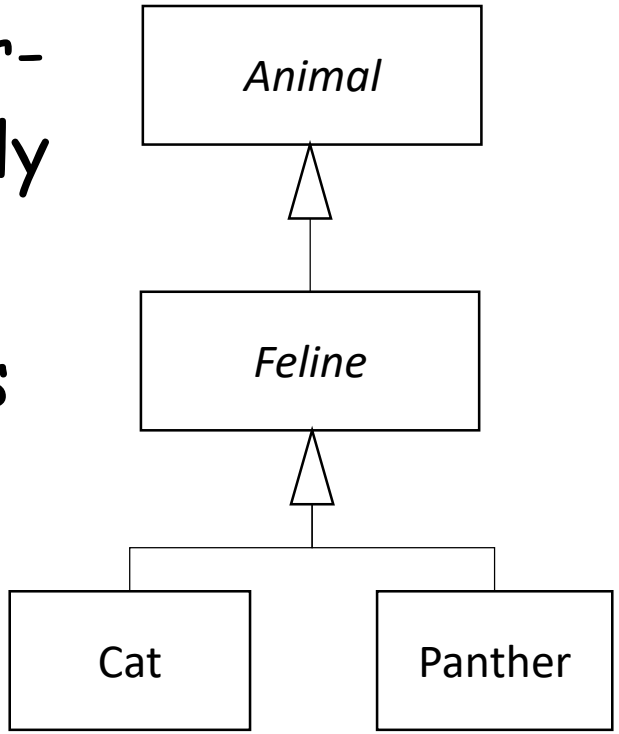
Questions?

- Purpose of constructors
- Default constructor

Constructor and Inheritance

- When we create an object of a class, constructors of all super-classes must be called explicitly or implicitly
- Can you name the constructors being called for this example?

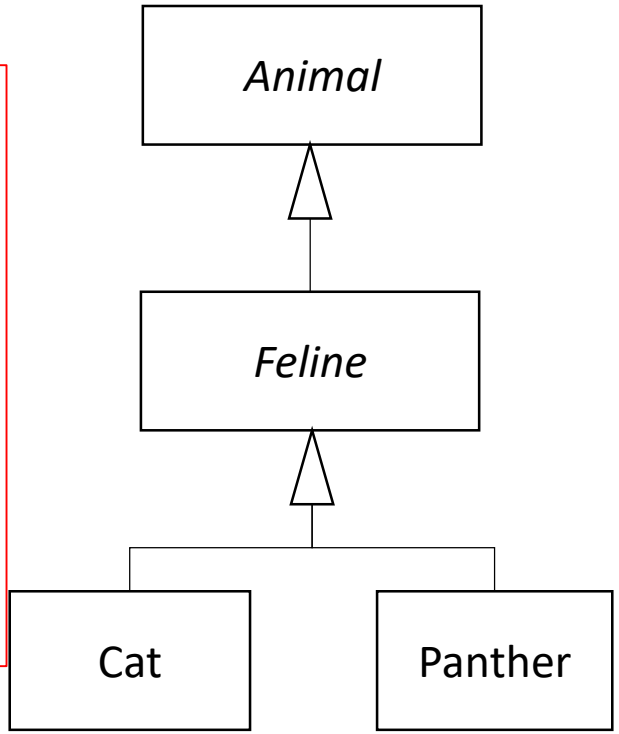
```
Panther brave = new Panther("brave");
```



Calling Super Class's Constructor Implicitly

- What if we write the constructor as follows,

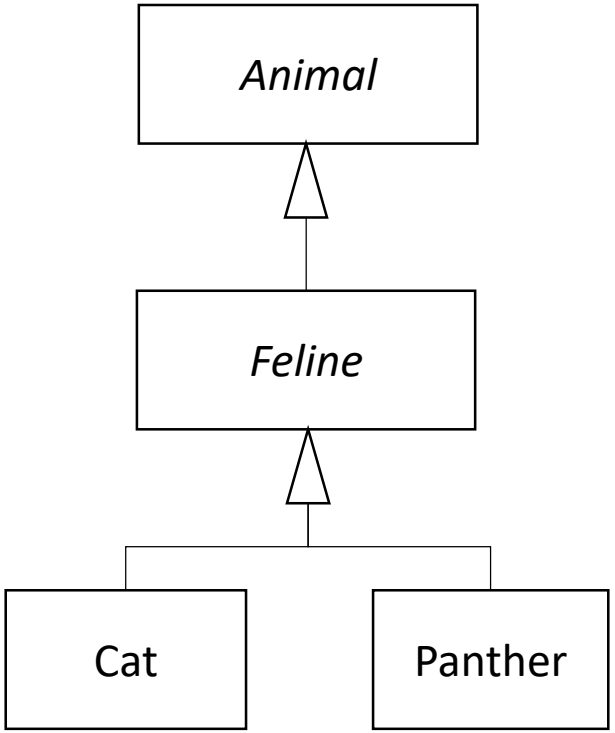
```
class Panther extends Feline {
   Color color;
   public Panther(String name, Color color) {
      this.color = color;
   }
   public void makeNoise() {…}
}
```



Calling Super Class's Constructor Implicitly

- Java compiler will call Feline's default constructor.

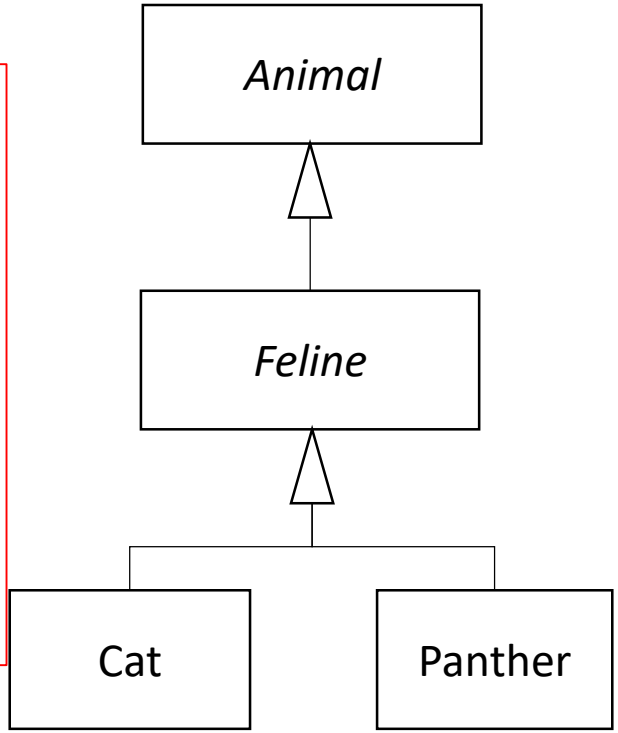
```
class Panther extends Feline {
   Color color;
   public Panther(String name, Color color) {
      this.color = color;
   }
   public void makeNoise() {…}
}
```



Calling Super Class's Constructor Explicitly

- Use "super"

```
class Panther extends Feline {
   Color color;
   public Panther(String name, Color color) {
        super(name);
        this.color = color;
   }
   public void makeNoise() {…}
}
```



Questions?

- Constructors
- Default constructor
- Overloading constructors
- Inheritance and constructors
- Stack and heap

Assignments

- Project 1
  - How is it going?
- CodeLab
- Upcoming Project 2
  - This Wednesday (on inheritance & polymorphism)