Object Oriented Programming

Class and object

- Remember this?

```
public class HelloWorld
{
        public static void main(String[] args)
        {
                System.out.println("HELLO CompSci 201!");
        }
}
```

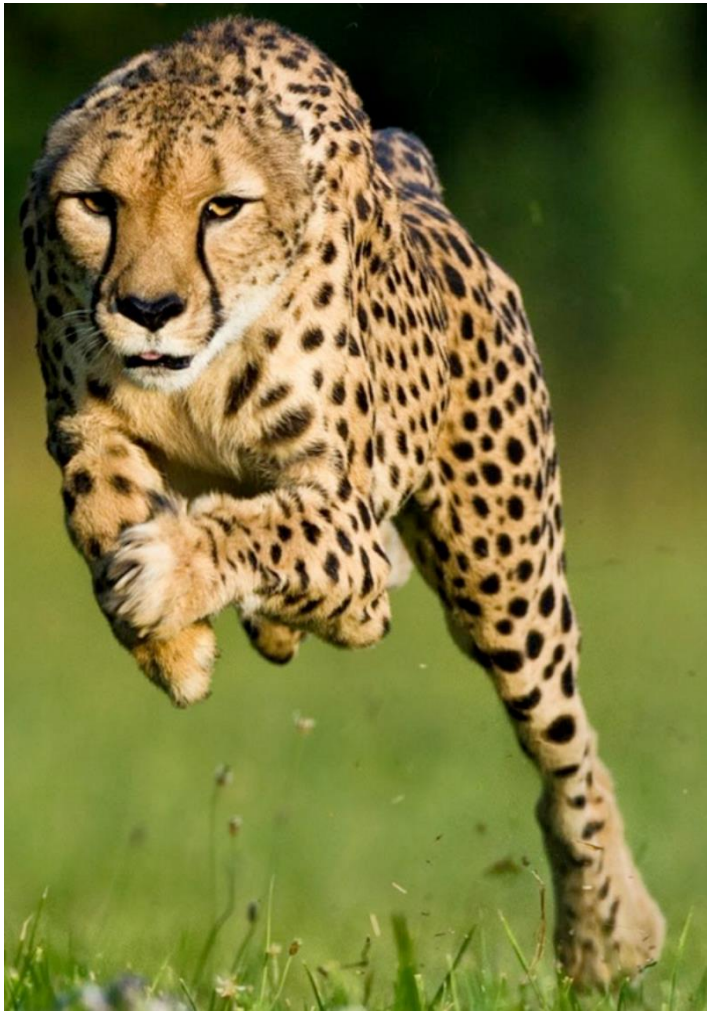
Class and object

- And this sounds familiar?
  - Array variables are object variables, so

```
// this won't work, because … ??

float myFloatArr[];
myFloatArray[0] = 5.5;
```

```
// what you must do is …

float myFloatArr = new float[3];
myFloatArr[0] = 5.5;
```

Objects in the world



- Are we ourselves objects as well ?

Objects in the world

- How to group the following objects ?

Animal

Objects in the world

- How to group the following objects ?





Insect

Objects in the world

- How to group the following objects ?

Transportation

Objects in the world

- How to group the following objects ?

→ **SportsCar**

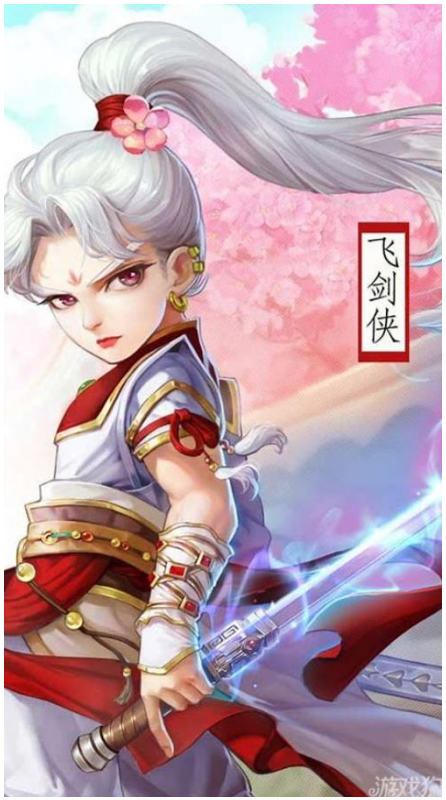
Objects in the world

- How to group the following objects ?

ComputerGame

Objects in the world

- How to group the following objects ?

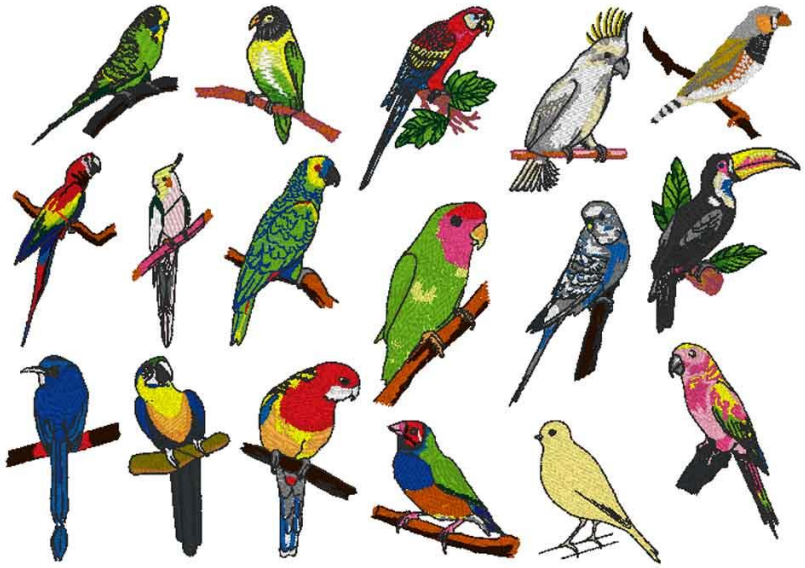


GameCharacter

- We do something similar to objects in the world of software …

Class and Object

- Similar objects can be grouped
  - We call this ‘group’ – class
  - And define it as ‘class Bird’



Bird

One more example here … we may group them as a class …

Vegetable

Class and Object

- Objects of a class share same attributes and behavior:

All birds have common properties:

name, species, age, color, feather, food (to eat), etc.

All birds may do common things:

fly, eat, perch, prey, etc.

Class and Object

- All properties (also called attributes) have a '(data) type' associated with them.
  - 'name' is a String, 'age' is an integer, 'food' is a type of 'FOOD' (defined somewhere), 'feather' is a type of FEATHER, …
    - FOOD and FEATHER are classes themselves!

```
All birds have attributes:

name (String)
age (int)
feather (FEATHER)
food (FOOD)
… …
```

```
All foods have attributes:

name (String)
calories (float)
weight (float)
originalPlace (String)
… …
```

Class and Object

- Actions (also called behaviors) may need something to work on, so called 'input/parameter', and these actions may then have a consequence
  - these actions are called '*method*'
  - the consequence is called a type of return

All birds may do:

… …

All birds have (attributes):

name, species, age, feather, food, etc.

All birds may (behave):

fly, eat, perch, grow, gainWeight, …

Class and Object

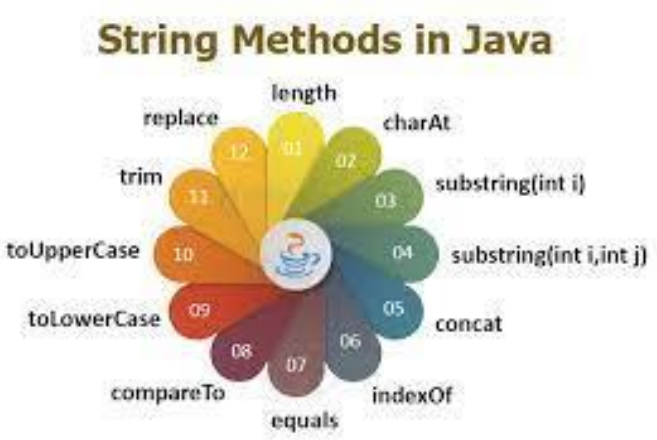
- Let's see another example – we can work together!

```
All computerGames have
   attributes:

… …
```

```
All computerGames have
   methods:

… …
```

Class and Object

- We now look at two Java built-in classes

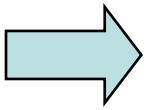


Class and Object

- And a class in Unity, a well-known game engine
  - Demo in Unity

Class and Object

- How to represent or model all these information?
  - Using C++, Java, or C#? Or better be language independent?



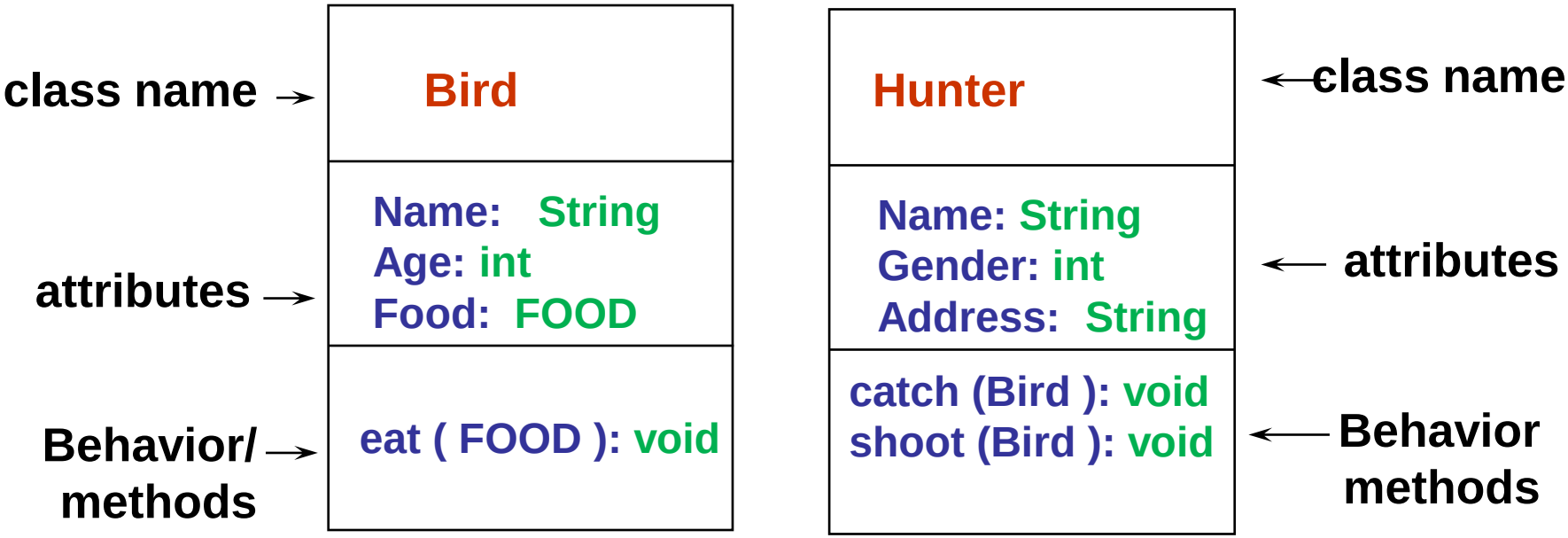
UML: Unified Modeling Language

UML is a standardized modeling language consisting of an integrated set of diagrams.

It is developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems.

Class and Object

- UML
  - Two examples here



Object-oriented programming in Java

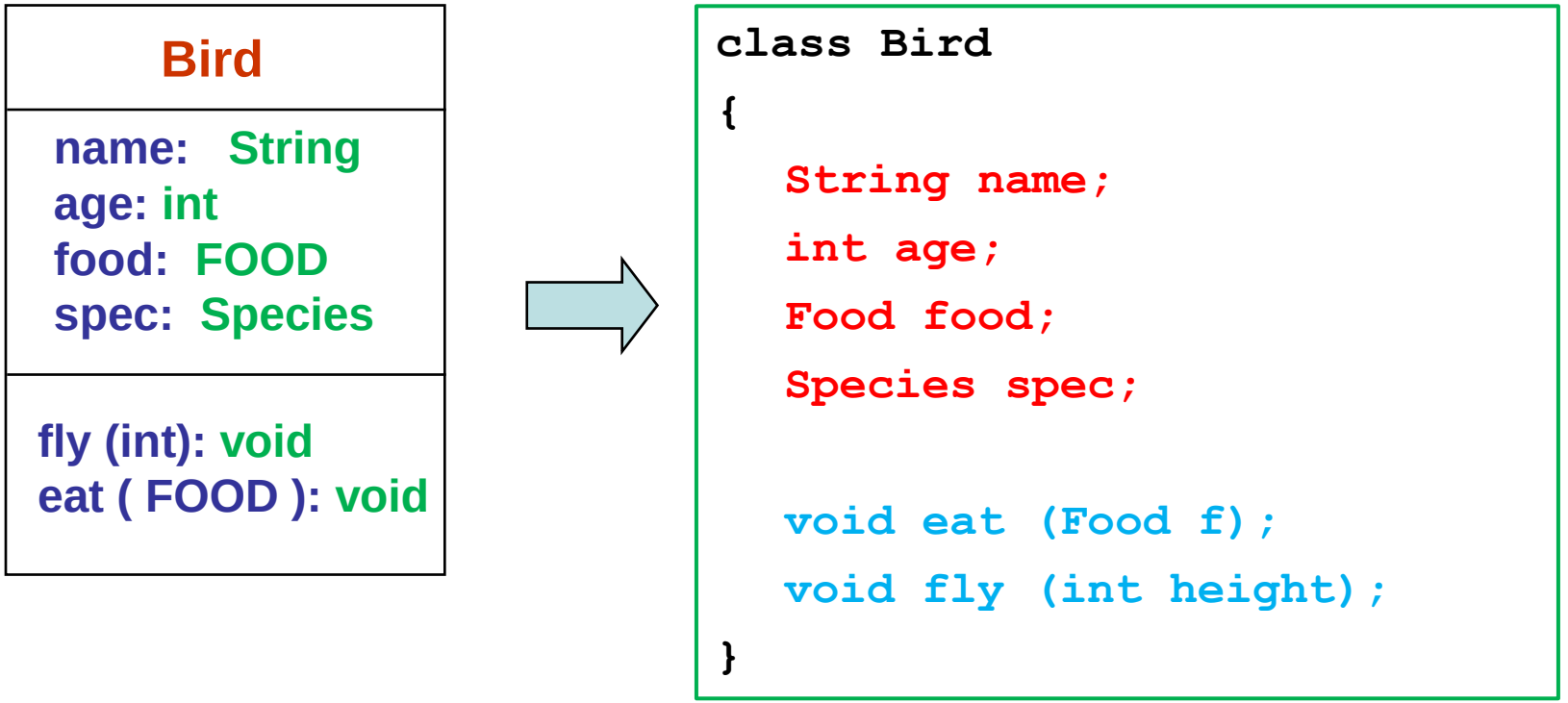
Class and Object

- We've already seen one …

```
public class HelloWorld
{
   public static void main(String []args)
   {
        System.out.println("Hello World");
   }
}
```

Class and Object

- Declare a class in Java



Class and Object

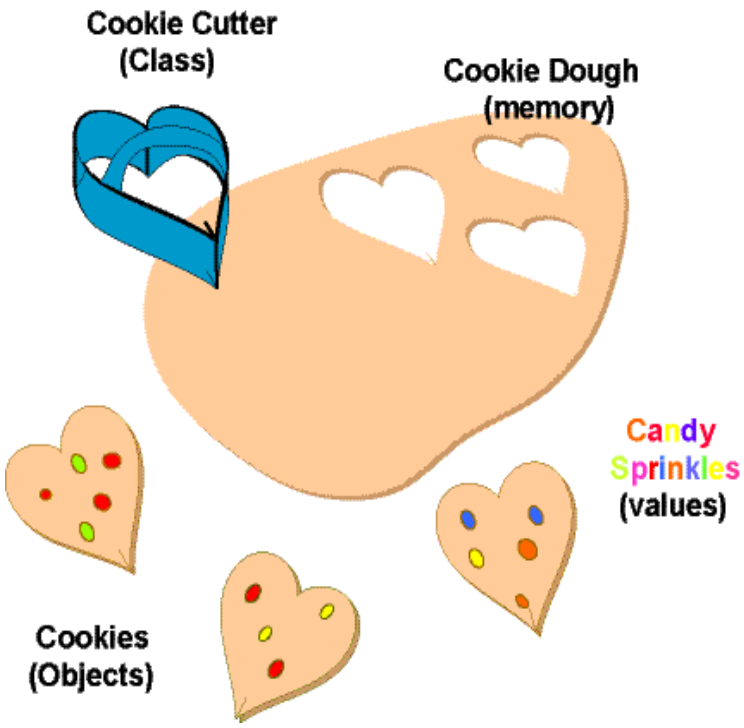
- What is an object?
  - An **instance** of a class, with its attributes clearly defined – objects differentiate themselves by their value of attributes

```
int main()
{
    Bird b1 = new Bird();
    b1.name = "coolGentleman";
    b1.age = 35;

    Bird b2 = new Bird();
    b2.name = "prettyLady";
    b2.age = 20;
}
```

```
class Bird
{
   public String name;
   public int age;
   Food food;
   Species spec;

   void eat (Food f);
   void fly (int height);
}
```

Class and Object

- Relationship of class and object – does the image on the right give you a hint?
  - ‘class’ acts like a template for creating (instantiating) ‘objects’ with same set of attributes (and methods).
  - ‘class’ is a blueprint, a design, something on paper, while ‘object’ is actual thing (e.g. a building), something concrete



Class and Object - Method

- Retrieve or manipulate an object's attributes, so called 'accessors':

  string getName( );

  void setName(string name);

- Define an object's behavior:

  void eat (Food f);

  void fly (int height);

```
class Bird
{
   String name;
   int age;
   Food food;
   Species spec;

   String getName ( );
   void setName(string s);

   void fly (int height);
   void eat (Food f);
}
```

Class and Object - Method

- Let's ***have a demo*** about attributes and methods, and how to use them.

```
class Bird
{
    String name;
    int age;
    Food food;
    Species spec;

    String getName ( );
    void setName(string s);

    void fly (int height);
    void eat (Food f);
}
```

```
// demo 1: define the methods in the class Bird

// demo 2: use the bird class and its methods
public class myBird
{
    Bird b1; // declare
    b1 = new Bird(); // define and initialize
    String name1 = b1.getName();
    … …

    // pitfall: use it before initialization
    Bird b2;
    String name2 = b2.getName();
}
```

Class and Object - Method

- Two special methods for a class:

  - Constructor:
    - has same name as the class
    - has no return type, not even 'void'

      void Bird (string, int) { … }

    - how many constructors may a class have?

```
class Bird {
   public String name;
   public int age;
   …
   Bird(String name, int
   age)
   {
      this.name = name;
      this.age = age;
   }
   void eat (Food f);
   …
}
```

Class and Object - Method

- Constructor:
  - Can have more than one, **multiple**!
  - They are *overloaded*
    - same name, but different parameters/arguments
  - play the role of creation and initialization

```
int main()
{
    Bird b1 = new Bird();
    System.out.println(b1.name + b1.age);

    Bird b2 = new Bird("Peggy", 2);
    System.out.println(b2.name + b2.age);
}
```

```
class Bird {
    public String name;
    public int age;
    …
    Bird(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
    Bird ()
    {
        this.name = "Rob";
        this.age = 1;
    }
}
```

Class and Object - Method

- Constructor:
  - must you (as programmer) provide one for your class?
  - automatically called whenever a new object of this class is created
  - so, a default constructor will be called if you have NOT provided any.

```
class Bird
{
   public String name;
   public int age;
   Food food;
   Species spec;

   void eat (Food f);
   void fly (int
   height);
}
```

```
int main()
{
      Bird b1 = new Bird();
      System.out.println(b1.name + b1.age);

      Bird b2 = new Bird();
}
```

Class and Object

- Instance and class variables/methods
  - Remember the ‘HelloWorld.class’?
  - What does the word ‘static’ mean here?

```
public class HelloWorld
{
   public static void main(String []args)
   {
         System.out.println("Hello World");
   }
}
```

Class and Object

- Instance variable/method:
  - represent the data associated with an object of a class
  - Each object of a class has a set of instance variables/methods
    - The birds, b1 and b2, each has its own age, name and the method fly

```
class Bird
{
   public String name;
   public int age;
    … …
   void fly (int
   height);
}
```

```
int main()
{
      Bird b1 = new Bird();
      b1.age = 2;

      Bird b2 = new Bird();
      b2.age = 3;
}
```

Class and Object

- Class (static) variable/method:
  - represent the data associated with a class, not any instance of the class
  - Designated with the key word '***static***'
  - each object of a class has or shares these static variables or methods.
    - The birds, b1 and b2, share the variable/attribute

```
class Bird {
    public String name;
    public int age;
    public static int numOfBirdsCreated = 0;
      … …
    Bird() {
          numOfBirdsCreated ++;
    }
    void fly (int height);
    public static int getNumOfBirdsCreated {
          Return numOfBirdsCreated;
    }
}
```

Class and Object

- Class variable/method:
  - each object of a class has or shares these static variables or methods.
    - The birds, b1 and b2, share 'numOfBirdsCreated'.

```
class Bird {
   public String name; public int age;
   public static int numOfBirdsCreated = 0;
    … …
   Bird() {
        numOfBirdsCreated ++;
   }
   void fly (int height);
   public static int getNumOfBirdsCreated {
        Return numOfBirdsCreated;
   }
}
```

```
int main()
{
     Bird b1 = new Bird();
     …
     Bird b2 = new Bird();
     …

     System.out.println("The
        number of birds created
        is " +

        Bird.numOfBridsCreate
        d);
}
```

Class and Object

- Class variable/method:
  - Any problem with the following code?

```
class Bird {
   public String name; public int age;
   public static int numOfBirdsCreated = 0;
     … …
   Bird() { numOfBirdsCreated ++;     }
   …
   public static int getNumOfBirdsCreated {
        return numOfBirdsCreated;
   }

   public static void printInfo() {
        System.out.println(name);
   }
}
```

Class and Object

- Class variable/method:
  - Any problem with the following code?

```
class Bird {
   public String name; public int age;
   public static int numOfBirdsCreated = 0;
     … …
   Bird() { numOfBirdsCreated ++;     }
   public static int getNumOfBirdsCreated {
        return numOfBirdsCreated;
   }
   public static void printInfo() {
        System.out.println(name);
   }
}
```

  - Error message: “non-static variable *name* cannot be referenced from a static context” !!

Class and Object

- Class variable/method:
  - A class/static variable can be accessed by both class/static and instance methods
    - The static variable belong to the class and is shared by all instances of the class, so everyone can access
  - An instance variable can NOT be accessed by static methods
    - The variable belongs to a particular instance

```
class Bird {
   public String name; public int age;
   public static int numOfBirdsCreated = 0;
   public static int getNumOfBirdsCreated {
        Return numOfBirdsCreated;
   }
   static void printInfo() { System.out.println(name); }
   void nonStaticPrint(){
        System.out.println(numOfBirdsCreated);
   }
}
```

Most important features of OO programming

- encapsulation
- inheritance
- polymorphism

Class and Object – Question?

- Is this .cpp compiable? Or any compilation error?
  - *'Bird::name': cannot access private member declared in class 'Bird'*
  - By default, attributes and methods are 'private' !!

```
int main()
{
    Bird b1 = new Bird();
    System.out.println(b1.name + b1.age);

    Bird b2 = new Bird("Peggy", 2);
    System.out.println(b1.name + b1.age);
}
```

```
class Bird {
    String name;
    int age;
    …
    Bird(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
    Bird ()
    {
        this.name = "Rob";
        this.age = 1;
    }
};
```

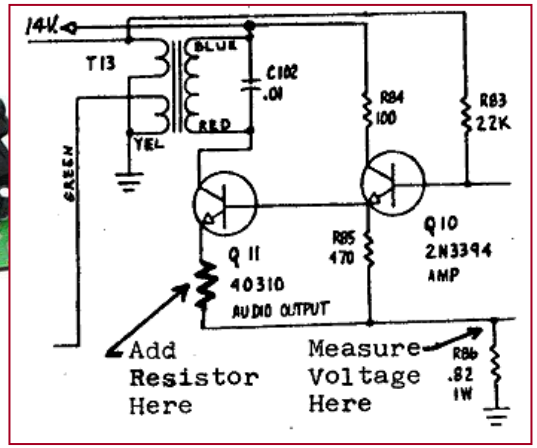
Encapsulation

- Providing access to an object only through its messages, while keeping details hidden – **black box**

- **Black box**
  - **Primary rule of OO programming is : as the user of an object, you should never need to peek inside the box, as the communication is done via messages.**

Encapsulation

- Hiding implementation details from clients
  - separates external view (behavior) from internal view (state)
  - protects the integrity of an object's data



Encapsulation

- Protects object from unwanted access
  - Example: Can't fraudulently increase an `Account`'s balance.
- Can change the class implementation later
  - Example: a class `Point` could be rewritten in polar coordinates (r, θ) with the same methods.
- Can constrain objects' state (**invariants**)
  - Example: Only allow `Account`s with non-negative balance.
  - Example: Only allow `Date`s with a month from 1-12.

Encapsulation

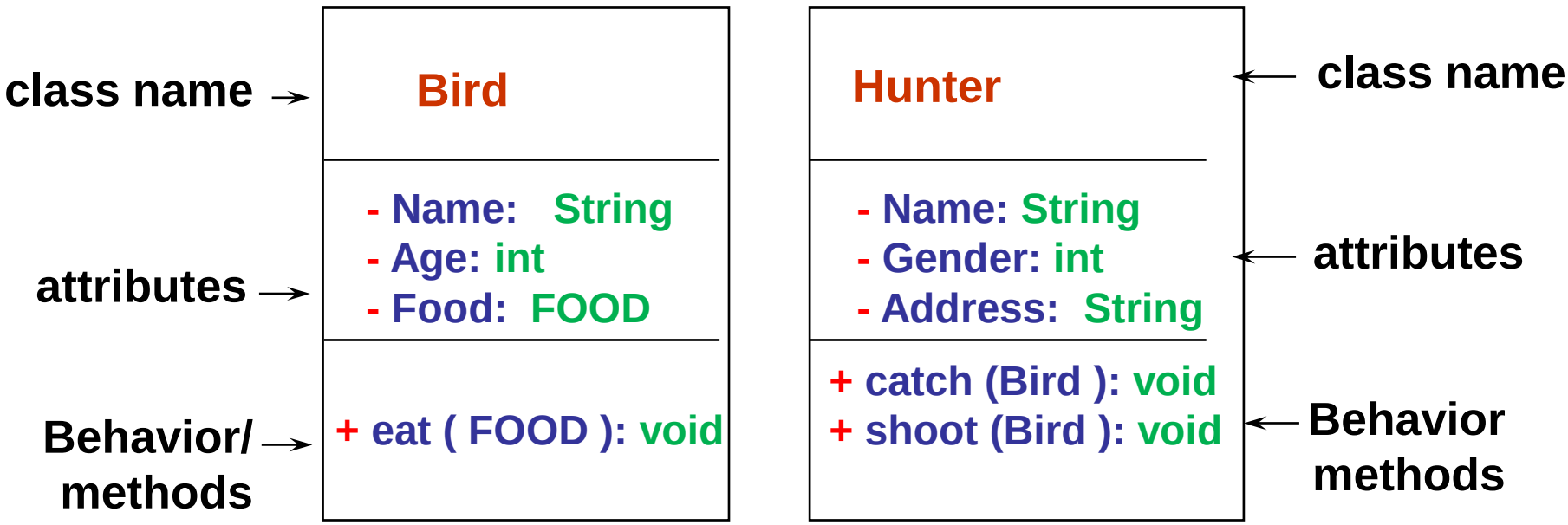
- Accessing a class's attributes (or data) through the class's methods (or interfaces)
  - 'private' for attributes
  - 'public' for methods

```
int main()
{
    Bird b1 = new Bird("Peggy", 6);
    System.out.println(b1.name +
        b1.age);

    System.out.println(b1.getName()
        );
}
```

```
class Bird {
    String name;
    int age;
    …
    Bird(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
    public string getName (){
        return name;
    }
    public int getAge() { … }
};
```

Encapsulation - UML

- How to represent different access level using UML?



- How about '*protected*' members?
  - Using the hash: #

Most important features of OO programming

- encapsulation
- inheritance
- polymorphism