Keywords `static`, and `final`

Shirley B. Chu

June 17, 2020

De La Salle University
College of Computer Studies

Instance Variables and Instance Methods

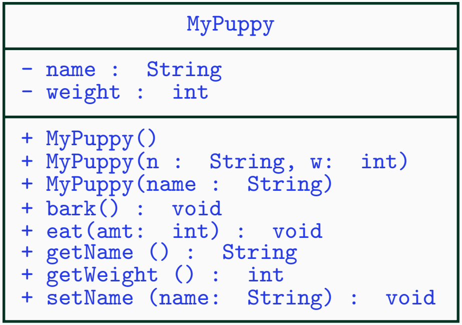
- *Instance variables* are variables declared inside the class. These variables (often called attributes) belong to each object of this class that we create.

Instance Variables and Instance Methods

- *Instance variables* are variables declared inside the class. These variables (often called attributes) belong to each object of this class that we create.
- *Instance methods* are methods declared inside the class. These methods belong to each object of this class that we create.

Instance Variables and Instance Methods

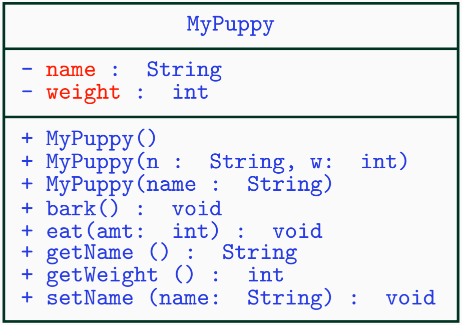
- *Instance variables* are variables declared inside the class. These variables (often called attributes) belong to each object of this class that we create.
- *Instance methods* are methods declared inside the class. These methods belong to each object of this class that we create.



```
MyPuppy
- name : String
- weight : int
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
```

Instance Variables and Instance Methods

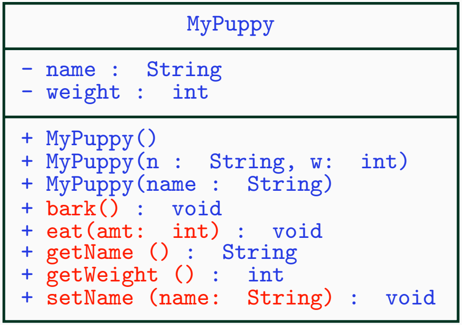
- *Instance variables* are variables declared inside the class. These variables (often called attributes) belong to each object of this class that we create.
- *Instance methods* are methods declared inside the class. These methods belong to each object of this class that we create.



```
MyPuppy
- name : String
- weight : int
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
```

Instance Variables and Instance Methods

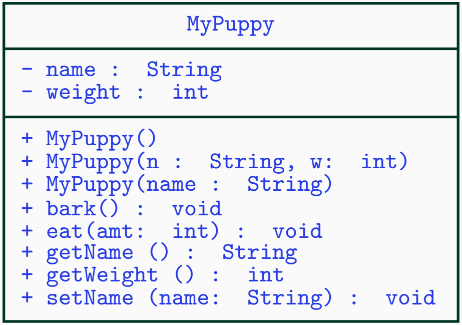
- *Instance variables* are variables declared inside the class. These variables (often called attributes) belong to each object of this class that we create.
- *Instance methods* are methods declared inside the class. These methods belong to each object of this class that we create.

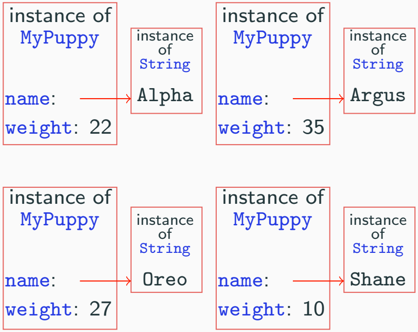


```
MyPuppy
- name : String
- weight : int
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
```

Instance Variables and Instance Methods

- *Instance variables* are variables declared inside the class. These variables (often called attributes) belong to each object of this class that we create.
- *Instance methods* are methods declared inside the class. These methods belong to each object of this class that we create.





Class Variables

- *Class variables* are declared as `static`.

Class Variables

- *Class variables* are declared as `static`.
- They are loaded in memory once the class is loaded.

Class Variables

- *Class variables* are declared as `static`.
- They are loaded in memory once the class is loaded.
- If visible, they may be accessed without creating an instance of the class, by using the class name.

Class Variables

- *Class variables* are declared as `static`.
- They are loaded in memory once the class is loaded.
- If visible, they may be accessed without creating an instance of the class, by using the class name.
- These variables are common to all objects of this class.

Class Variables

- *Class variables* are declared as `static`.
- They are loaded in memory <u>once the class is loaded</u>.
- If visible, they may be accessed without creating an instance of the class, by using the class name.
- These variables are common to all objects of this class.
- In the UML diagram, these are <u>underlined</u>.

Class Variables

```
MyPuppy
------------------------------------------
- name :  String
- weight :  int
------------------------------------------
+ MyPuppy()
+ MyPuppy(n :  String, w:  int)
+ MyPuppy(name :  String)
+ bark() :  void
+ eat(amt:  int) :  void
+ getName () :  String
+ getWeight () :  int
+ setName (name:  String) :  void
```

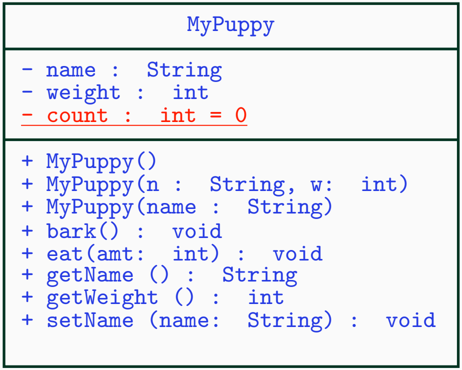
```
public class MyPuppy
{

    private String name;
    private int weight;
            ⋮
}
```

Class Variables

```
MyPuppy
-----------------------------------------------
- name : String
- weight : int
- count : int = 0
-----------------------------------------------
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
```

```
public class MyPuppy
{

    private String name;
    private int weight;
            ⋮
}
```

Class Variables



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
```

```
public class MyPuppy
{
    private static int count = 0;
    private String name;
    private int weight;
              ⋮
}
```

Class Methods

- *Class methods* are declared as `static`.

Class Methods

- *Class methods* are declared as `static`.
- They are loaded in memory once the class is loaded.

Class Methods

- *Class methods* are declared as `static`.
- They are loaded in memory once the class is loaded.
- If visible, they may be invoked without creating an instance of the class, by using the class name.

Class Methods

- *Class methods* are declared as `static`.
- They are loaded in memory once the class is loaded.
- If visible, they may be invoked without creating an instance of the class, by using the class name.
- These methods are common to all objects of this class.

Class Methods

- *Class methods* are declared as `static`.
- They are loaded in memory <u>once the class is loaded</u>.
- If visible, they may be invoked without creating an instance of the class, by using the class name.
- These methods are common to all objects of this class.
- In the UML diagram, these are <u>underlined</u>.

Class Methods

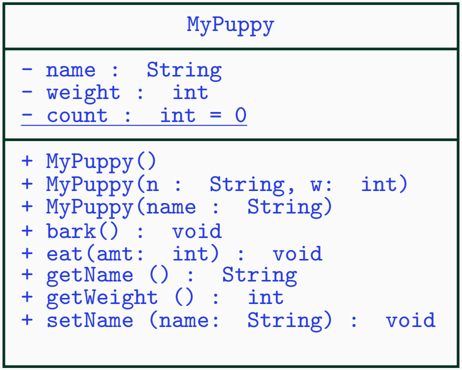
- *Class methods* are declared as `static`.
- They are loaded in memory <u>once the class is loaded</u>.
- If visible, they may be invoked without creating an instance of the class, by using the class name.
- These methods are common to all objects of this class.
- In the UML diagram, these are <u>underlined</u>.
- Class methods are used to access class variables.

Class Methods

- *Class methods* are declared as `static`.
- They are loaded in memory <u>once the class is loaded</u>.
- If visible, they may be invoked without creating an instance of the class, by using the class name.
- These methods are common to all objects of this class.
- In the UML diagram, these are <u>underlined</u>.
- Class methods are used to access class variables.

Some rules to take note of ▸ Rules

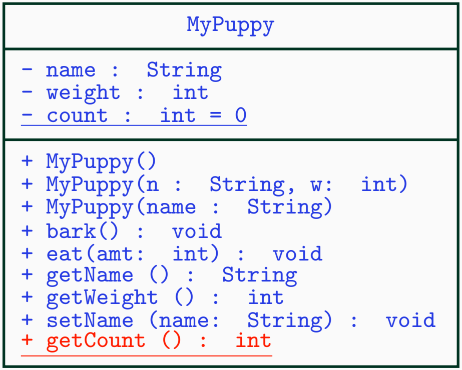
Class Methods



```
MyPuppy

- name : String
- weight : int
- count : int = 0

+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
```

```
public class MyPuppy
{
    private static int count = 0;
    private String name;
    private int weight;
            ⋮

}
```

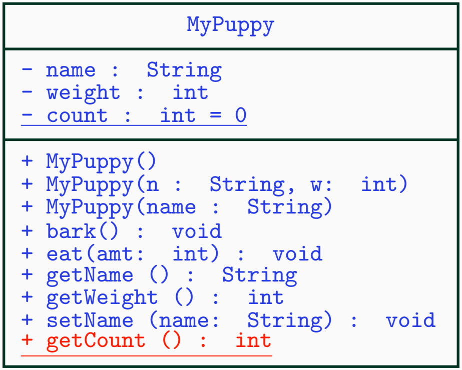
Class Methods



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```
public class MyPuppy
{
    private static int count = 0;
    private String name;
    private int weight;
            ⋮
}
```

Class Methods



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

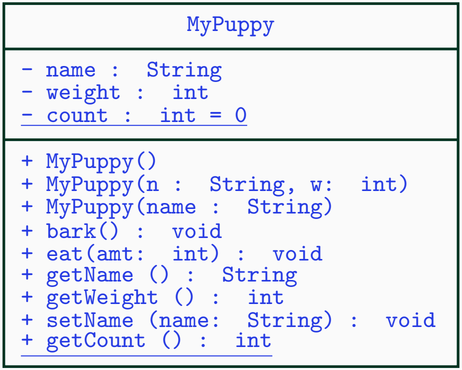
```
public class MyPuppy
{
    private static int count = 0;
    private String name;
    private int weight;
              :
    public static int getCount ()
    {
        return count;
    }
}
```

Using class variables

```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```
public class MyPuppy
{
    public MyPuppy ()
    {   name = "puppy";
        weight = 10;
    }
    public MyPuppy (String name)
    {   this ();
        this.name = name;
    }

    public MyPuppy (String n, int w)
    {   name = n;
        this.weight = w;
    }
}
```

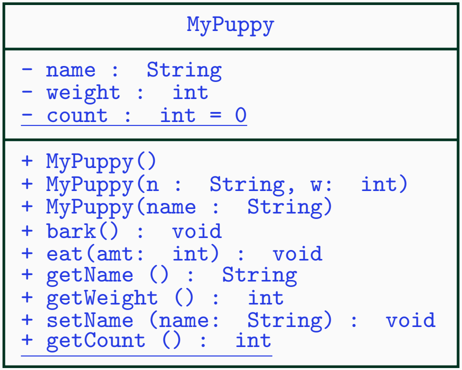
Using class variables



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```java
public class MyPuppy
{
    public MyPuppy ()
    {   name = "puppy";
        weight = 10;
        count++;
    }
    public MyPuppy (String name)
    {   this ();
        this.name = name;
    }

    public MyPuppy (String n, int w)
    {   name = n;
        this.weight = w;
    }
}
```

Using class variables



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```java
public class MyPuppy
{
    public MyPuppy ()
    {   name = "puppy";
        weight = 10;
        count++;
    }
    public MyPuppy (String name)
    {   this ();
        this.name = name;
    }

    public MyPuppy (String n, int w)
    {   name = n;
        this.weight = w;
        count++;
    }
}
```

The keyword final

- Constants can be defined by declaring it `final`, e.g.

```
public final int SIZE = 30;
```

The keyword `final`

- Constants can be defined by declaring it `final`, e.g.

```
public final int SIZE = 30;
```

- Naming convention for constants: nouns or noun phrases; all uppercase letters separated by underscores _

The keyword `final`

- Constants can be defined by declaring it `final`, e.g.

  ```
  public final int SIZE = 30;
  ```
- Naming convention for constants: nouns or noun phrases; all uppercase letters separated by underscores _
- In the UML diagram, these are encoded in all uppercase letters

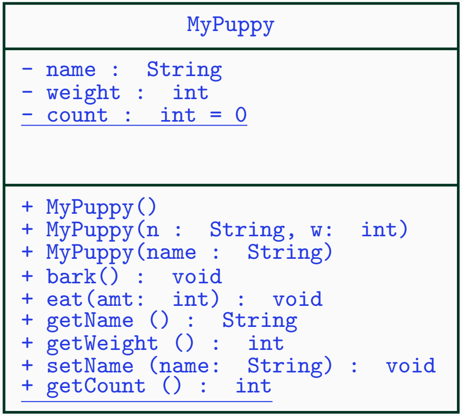
The keyword `final`

- Constants can be defined by declaring it `final`, e.g.

```
public final int SIZE = 30;
```

- Naming convention for constants: nouns or noun phrases; all uppercase letters separated by underscores _
- In the UML diagram, these are encoded in all uppercase letters

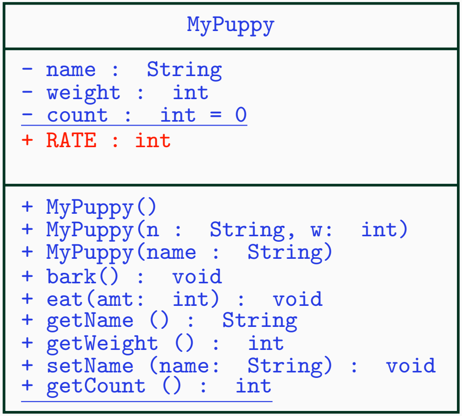
Class Variables



```
MyPuppy

- name : String
- weight : int
- count : int = 0

+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```
public class MyPuppy
{
    private static int count = 0;
    private String name;
    private int weight;
              ⋮
}
```

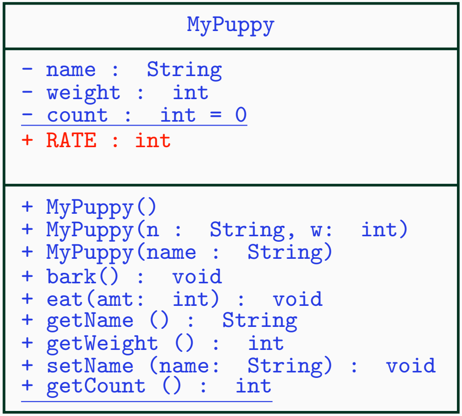
Class Variables



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ RATE : int

+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```
public class MyPuppy
{
    private static int count = 0;
    private String name;
    private int weight;
            ⋮
}
```

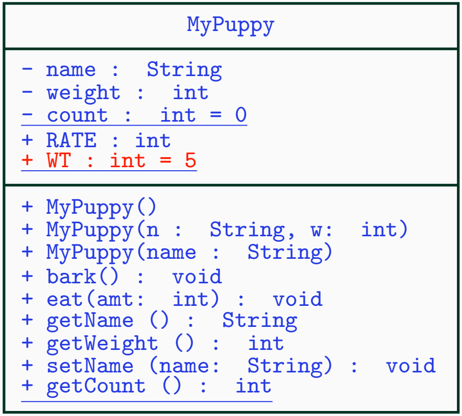
Class Variables



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ RATE : int

+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```
public class MyPuppy
{
    public final int RATE;
    private static int count = 0;
    private String name;
    private int weight;
            ⋮
}
```

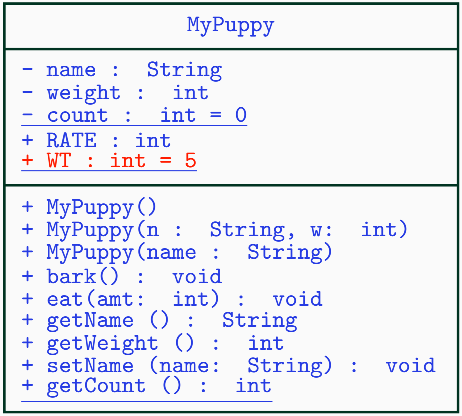
Class Variables



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ RATE : int
+ WT : int = 5
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```
public class MyPuppy
{
    public final int RATE;
    private static int count = 0;
    private String name;
    private int weight;
              ⋮
}
```

Class Variables



```
MyPuppy
- name : String
- weight : int
- count : int = 0
+ RATE : int
+ WT : int = 5
+ MyPuppy()
+ MyPuppy(n : String, w: int)
+ MyPuppy(name : String)
+ bark() : void
+ eat(amt: int) : void
+ getName () : String
+ getWeight () : int
+ setName (name: String) : void
+ getCount () : int
```

```
public class MyPuppy
{
    public final int RATE;
    public static final int WT = 5;
    private static int count = 0;
    private String name;
    private int weight;
              ⋮
}
```

Blank final

- A `final` variable can only be assigned a value <u>once</u>.

Blank final

- A `final` variable can only be assigned a value once.
- A *blank final* is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.

Blank final

- A `final` variable can only be assigned a value <u>once</u>.
- A *blank final* is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.
- Some points to remember:

Blank final

- A `final` variable can only be assigned a value once.
- A ***blank final*** is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.
- Some points to remember:
  - A blank final instance variable

Blank final

- A `final` variable can only be assigned a value <u>once</u>.
- A *blank final* is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.
- Some points to remember:
    - A blank final instance variable
        - must be initialize at the constructor;

Blank final

- A `final` variable can only be assigned a value once.
- A *blank final* is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.
- Some points to remember:
    - A blank final instance variable
        - must be initialize at the constructor;
        - cannot be initialize in class methods.

Blank final

- A `final` variable can only be assigned a value once.
- A *blank final* is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.
- Some points to remember:
  - A blank final instance variable
    - must be initialize at the constructor;
    - cannot be initialize in class methods.
  - A blank final class variable

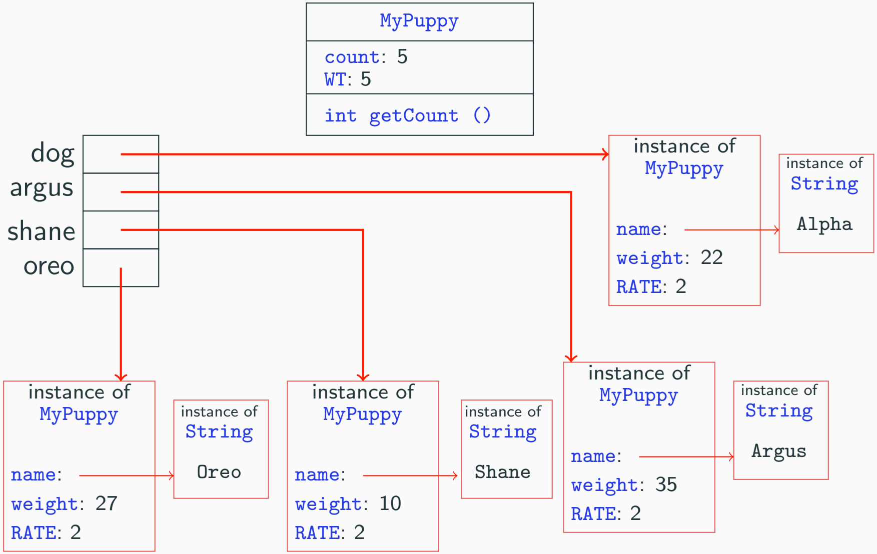
Blank final

- A `final` variable can only be assigned a value once.
- A *blank final* is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.
- Some points to remember:
  - A blank final instance variable
    - must be initialize at the constructor;
    - cannot be initialize in class methods.
  - A blank final class variable
    - cannot be left uninitialized.

Blank final

- A `final` variable can only be assigned a value once.
- A *blank final* is a `final` variable that has not been initialized during declaration. Its value may be assigned at a later stage.
- Some points to remember:
  - A blank final instance variable
    - must be initialize at the constructor;
    - cannot be initialize in class methods.
  - A blank final class variable
    - cannot be left uninitialized.
    - cannot be initialized in constructor or class methods.

Alpha, Argus, Oreo, and Shane



☺ **Thank you!** ☺

Class Members and Instance Members

- Instance methods can access instance variables and instance methods directly.

Class Members and Instance Members

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.

Class Members and Instance Members

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.

Class Members and Instance Members

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.
- Class methods cannot access instance variables or instance methods directly?they must use an object reference.

Class Members and Instance Members

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.
- Class methods cannot access instance variables or instance methods directly?they must use an object reference.
- Also, class methods cannot use the `this` keyword as there is no instance for `this` to refer to.

Class Members and Instance Members

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.
- Class methods cannot access instance variables or instance methods directly?they must use an object reference.
- Also, class methods cannot use the `this` keyword as there is no instance for `this` to refer to.

◂ Back