UNTAR
Universitas Tarumanagara

Terakreditasi BAN PT A (Unggul)

UNTAR untuk INDONESIA

Object Based Programming

825200008 - ALDI RESALDI MAULANA

825200049 - AFINA PUTRI DAYANTI

825200050 - ERIC ANTHONY

PROGRAM STUDI SISTEM INFORMASI
UNIVERSITAS TARUMANAGARA

Static & Non Static Variable

Static Variables

Static variables are, essentially, global variables. All instances of the class share the same static variable.

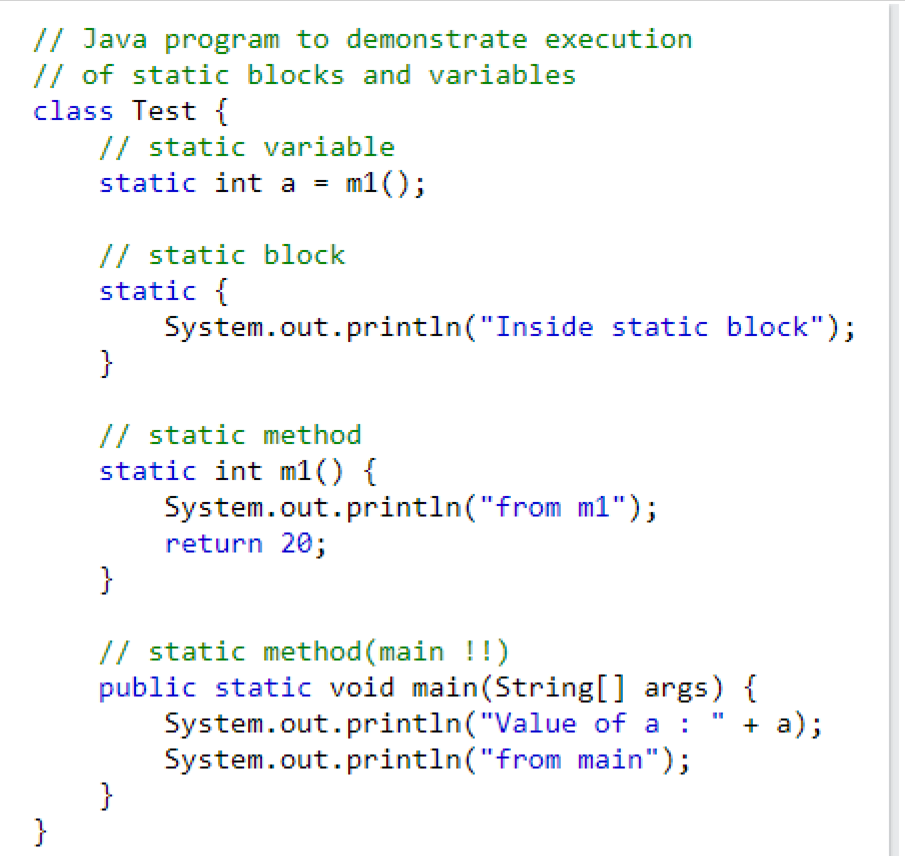
Non-Static Variable

- **Local Variables**: A variable defined within a block or method or constructor is called local variable.
- **Instance Variables:** Instance variables are non-static variables and are declared in a class outside any method, constructor or block.

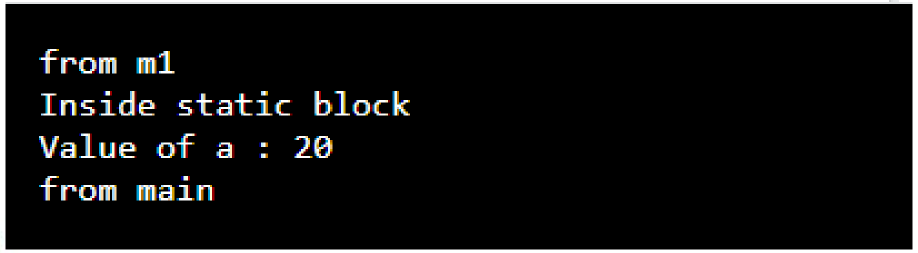
Static & Non Static Variable

Static variable	Non static variable
Static variables can be accessed using class name	Non static variables can be accessed using instance of a class
Static variables can be accessed by static and non static methods	Non static variables cannot be accessed inside a static method.
Static variables reduce the amount of memory used by a program.	Non static variables do not reduce the amount of memory used by a program
Static variables are shared among all instances of a class.	Non static variables are specific to that instance of a class.
Static variable is like a global variable and is available to all methods.	Non static variable is like a local variable and they can be accessed through only instance of a class.

Static Variable

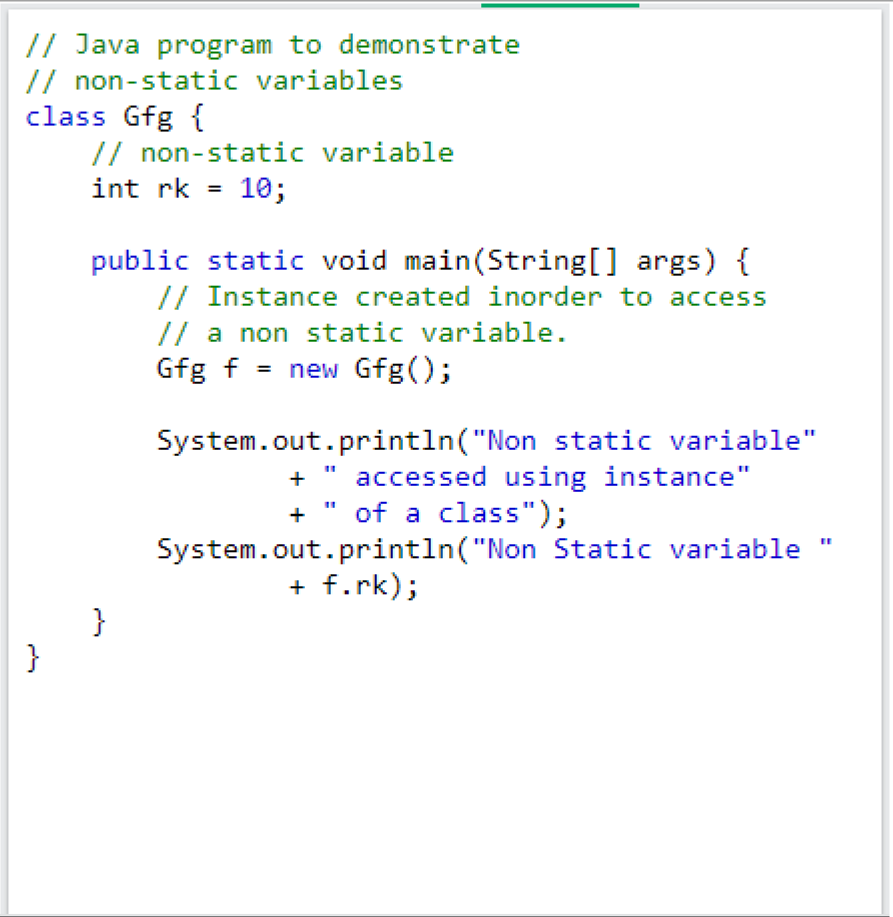


```java
// Java program to demonstrate execution
// of static blocks and variables
class Test {
    // static variable
    static int a = m1();

    // static block
    static {
        System.out.println("Inside static block");
    }

    // static method
    static int m1() {
        System.out.println("from m1");
        return 20;
    }

    // static method(main !!)
    public static void main(String[] args) {
        System.out.println("Value of a : " + a);
        System.out.println("from main");
    }
}
```

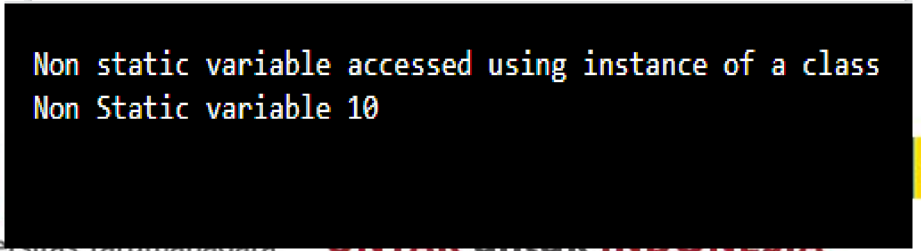


```
from m1
Inside static block
Value of a : 20
from main
```

Non-Static Method



```java
// Java program to demonstrate
// non-static variables
class Gfg {
    // non-static variable
    int rk = 10;

    public static void main(String[] args) {
        // Instance created inorder to access
        // a non static variable.
        Gfg f = new Gfg();

        System.out.println("Non static variable"
                + " accessed using instance"
                + " of a class");
        System.out.println("Non Static variable "
                + f.rk);
    }
}
```



```
Non static variable accessed using instance of a class
Non Static variable 10
```

Static & Non Static Method

Static Method

A **static method** is a method that belongs to a class, but it does not belong to an instance of that class and this method can be called without the instance or object of that class.

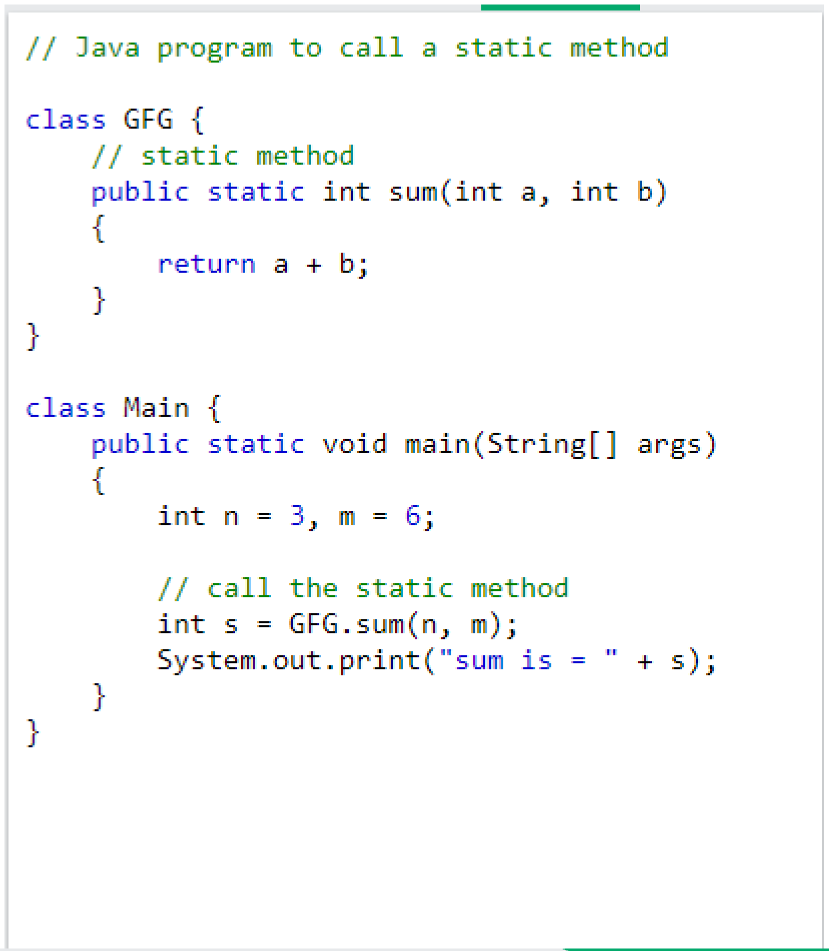
Non-Static Method

Every methods in java defaults to non-static method without **static** keyword preceding it. **non-static** methods can access any **static** method and **static** variable also, without using the object of the class.

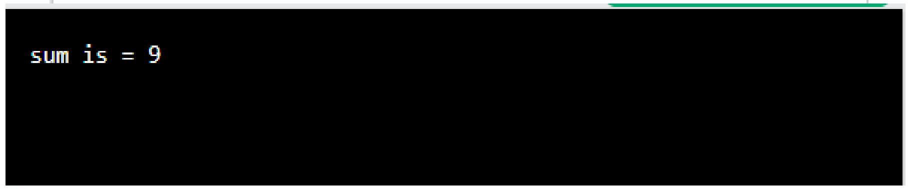
Static & Non Static Method

Points	static method	non-static method
Accessing members and methods	In **static** method, the method can only access only static data members and static methods of another class or same class but cannot access non-static methods and variables.	In **non-static** method, the method can access static data members and static methods as well as non-static members and method of another class or same class.
Binding process	**Static** method uses compile time or early binding.	**Non-static** method uses runtime or dynamic binding.
Overriding	**Static** method cannot be overridden because of early binding.	**Non-static** method can be overridden because of runtime binding.
Memory allocation	In **static** method, less memory is use for execution because memory allocation happens only once, because the static keyword fixed a particular memory for that method in ram.	In **non-static** method, much memory is used for execution because here memory allocation happens when the method is invoked and the memory is allocated every time when the method is called.

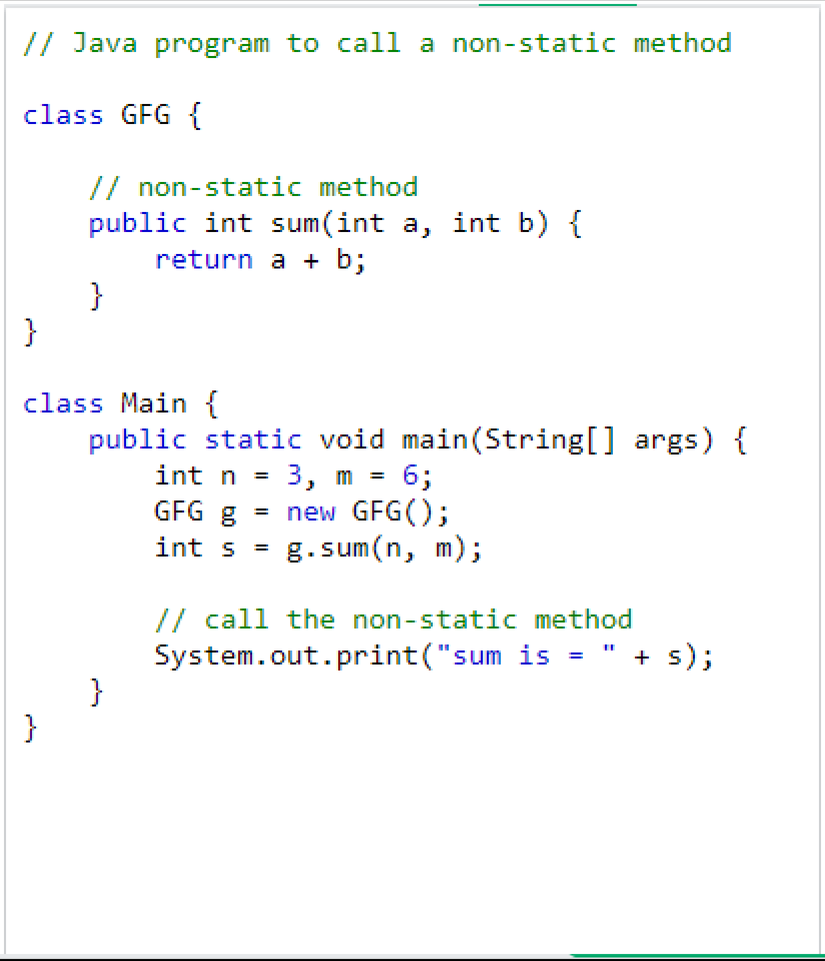
Static Method	Non-Static Method



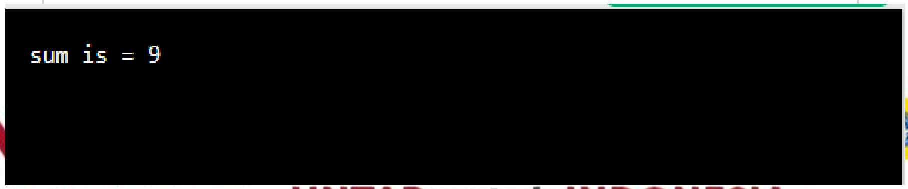
```java
// Java program to call a static method

class GFG {
    // static method
    public static int sum(int a, int b)
    {
        return a + b;
    }
}

class Main {
    public static void main(String[] args)
    {
        int n = 3, m = 6;

        // call the static method
        int s = GFG.sum(n, m);
        System.out.print("sum is = " + s);
    }
}
```



```
sum is = 9
```

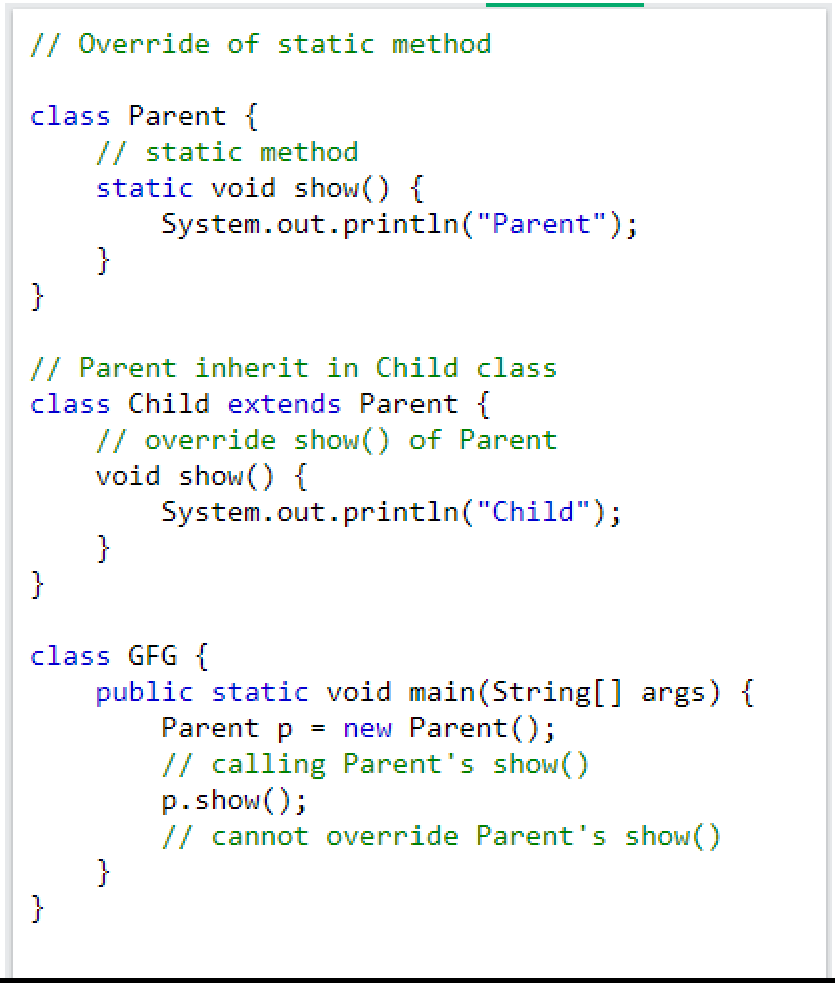


```java
// Java program to call a non-static method

class GFG {

    // non-static method
    public int sum(int a, int b) {
        return a + b;
    }
}

class Main {
    public static void main(String[] args) {
        int n = 3, m = 6;
        GFG g = new GFG();
        int s = g.sum(n, m);

        // call the non-static method
        System.out.print("sum is = " + s);
    }
}
```

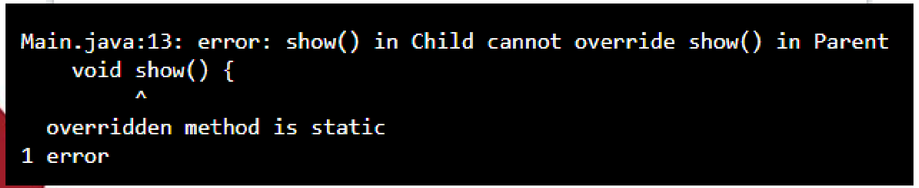


```
sum is = 9
```

Override Static Method

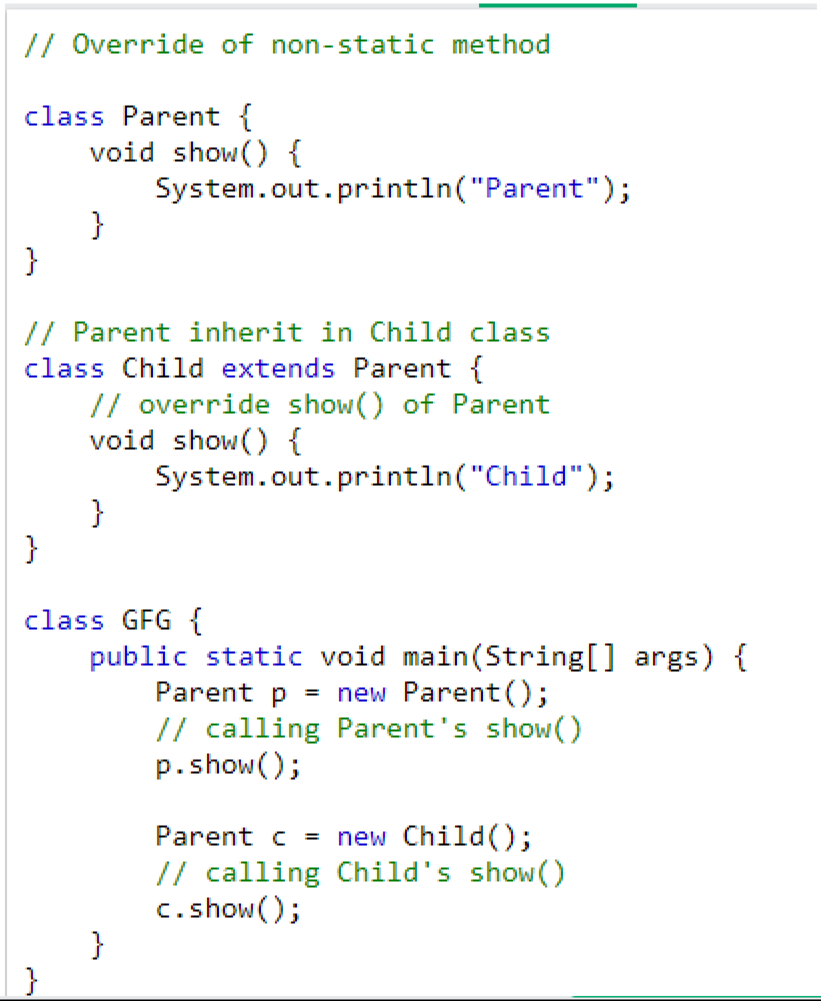


```java
// Override of static method

class Parent {
    // static method
    static void show() {
        System.out.println("Parent");
    }
}

// Parent inherit in Child class
class Child extends Parent {
    // override show() of Parent
    void show() {
        System.out.println("Child");
    }
}

class GFG {
    public static void main(String[] args) {
        Parent p = new Parent();
        // calling Parent's show()
        p.show();
        // cannot override Parent's show()
    }
}
```

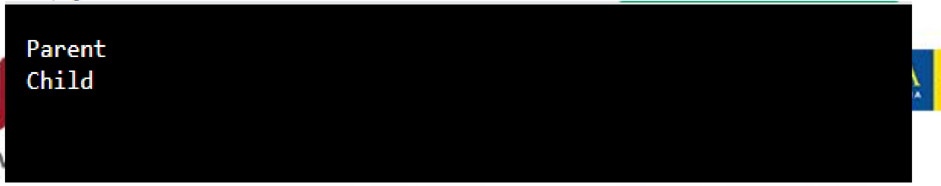


```
Main.java:13: error: show() in Child cannot override show() in Parent
    void show() {
         ^
  overridden method is static
1 error
```

Override Non-Static Method



```java
// Override of non-static method

class Parent {
    void show() {
        System.out.println("Parent");
    }
}

// Parent inherit in Child class
class Child extends Parent {
    // override show() of Parent
    void show() {
        System.out.println("Child");
    }
}

class GFG {
    public static void main(String[] args) {
        Parent p = new Parent();
        // calling Parent's show()
        p.show();

        Parent c = new Child();
        // calling Child's show()
        c.show();
    }
}
```



```
Parent
Child
```

Abstract

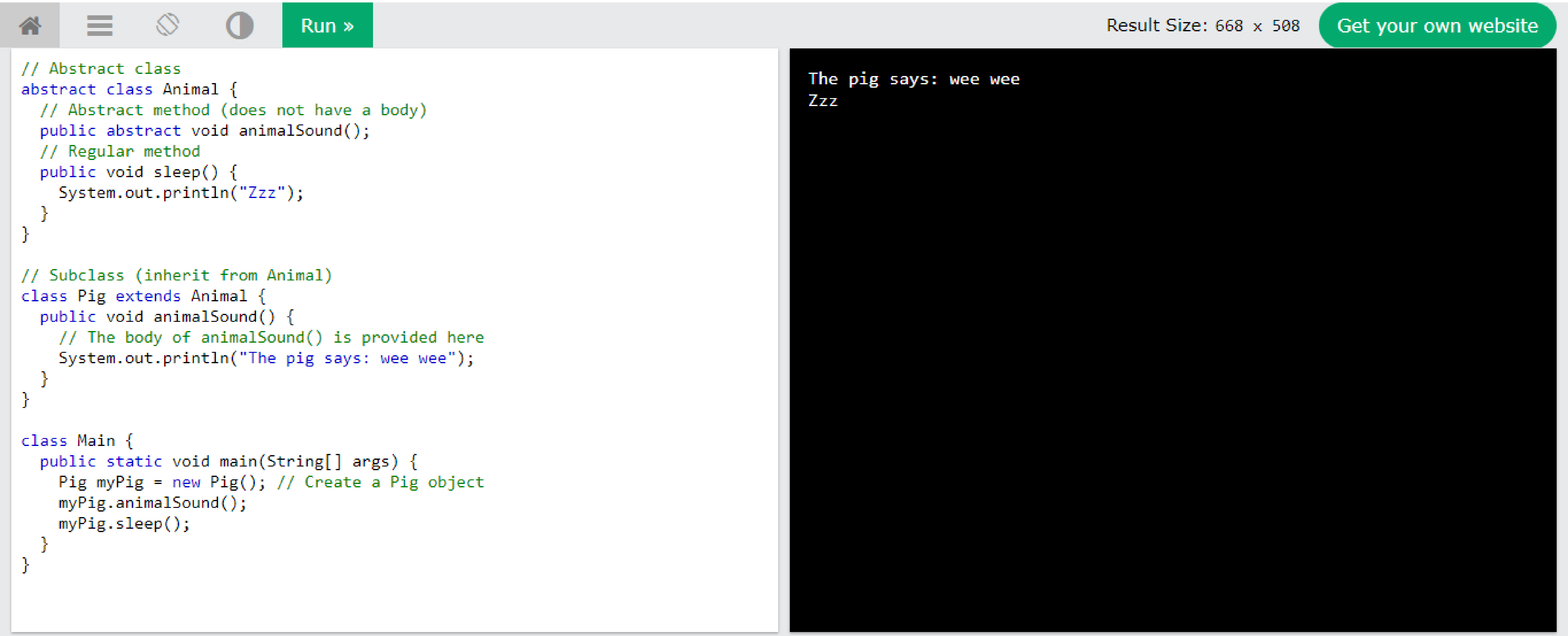
Abstract class

Abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method

Abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

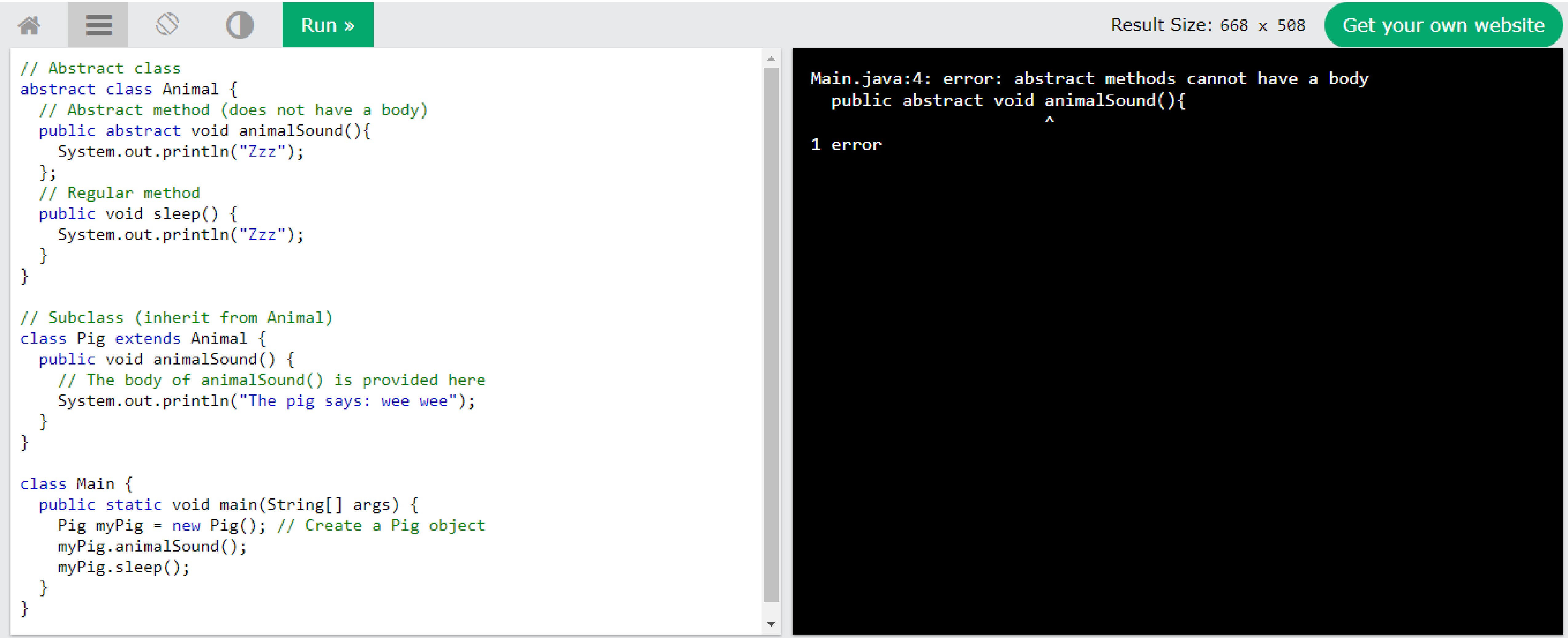
Correct example using abstract class



```java
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

```
The pig says: wee wee
Zzz
```

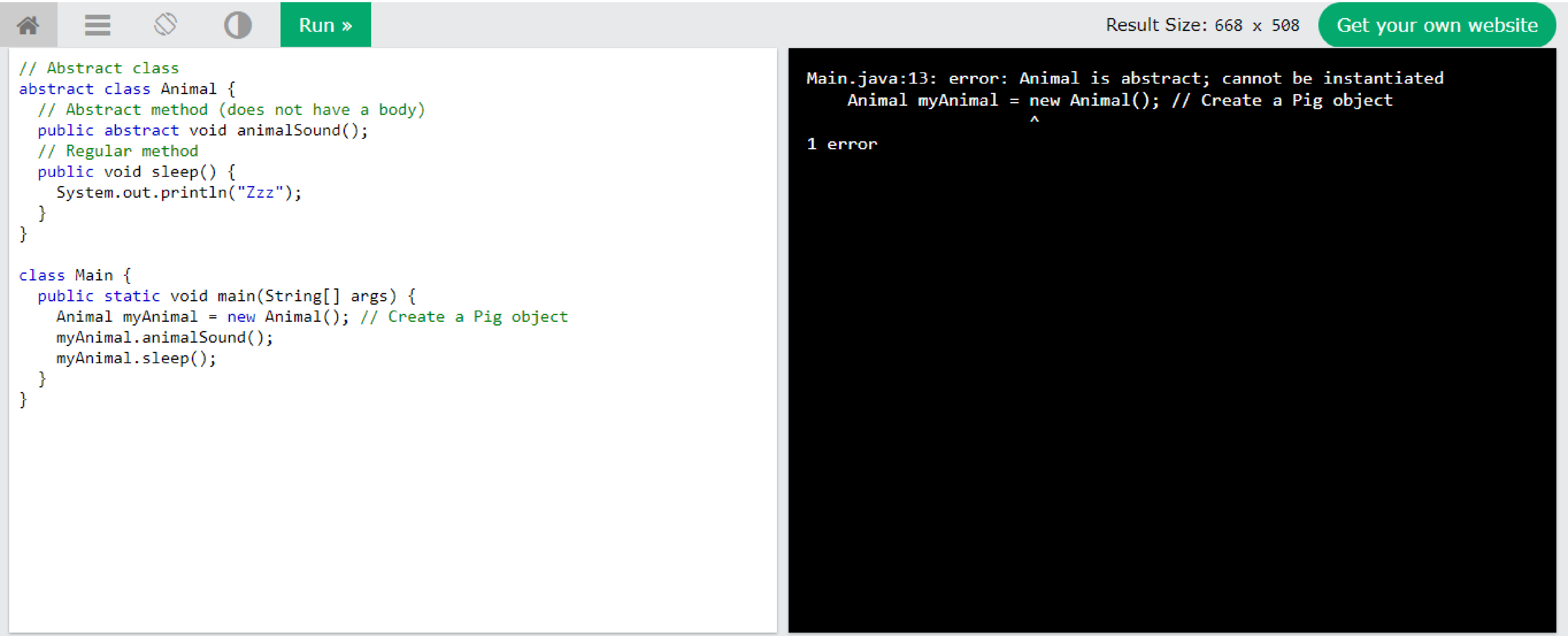
Incorrect example using abstract class



```java
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound(){
    System.out.println("Zzz");
  };
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

```
Main.java:4: error: abstract methods cannot have a body
  public abstract void animalSound(){
                     ^
1 error
```

How about directly calling the abstract class?



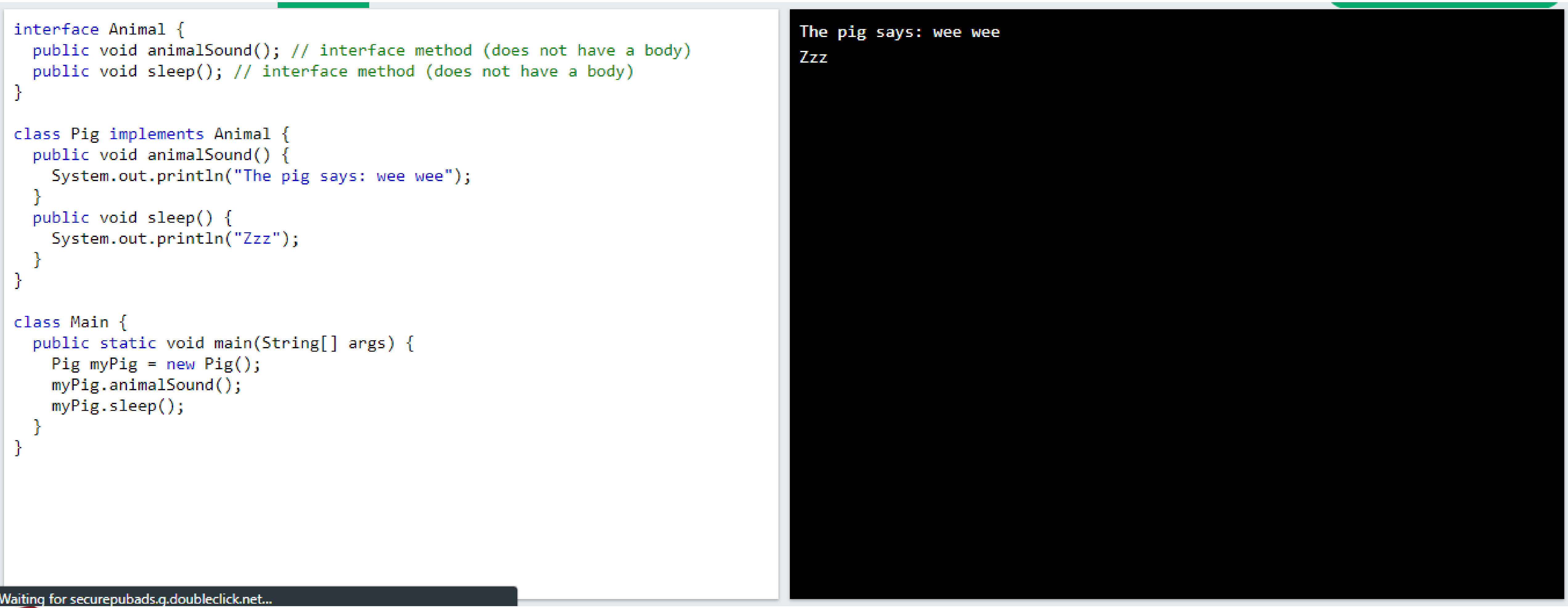
```java
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal(); // Create a Pig object
    myAnimal.animalSound();
    myAnimal.sleep();
  }
}
```

```
Main.java:13: error: Animal is abstract; cannot be instantiated
    Animal myAnimal = new Animal(); // Create a Pig object
                      ^
1 error
```

Interfaces

An interface is a completely "**abstract class**" that is used to group related methods with empty bodies. To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class.

Correct example using Interface

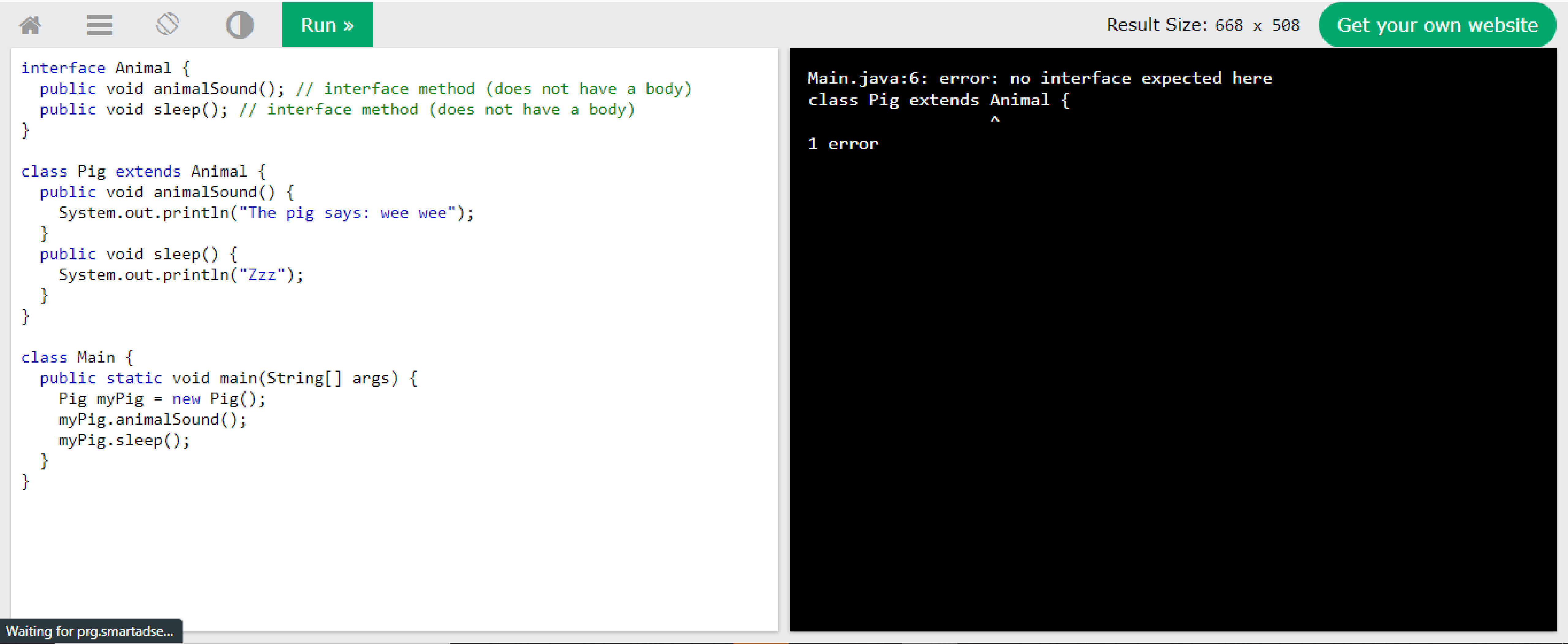


```
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

class Pig implements Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();
    myPig.animalSound();
    myPig.sleep();
  }
}
```

```
The pig says: wee wee
Zzz
```

Waiting for securepubads.g.doubleclick.net...

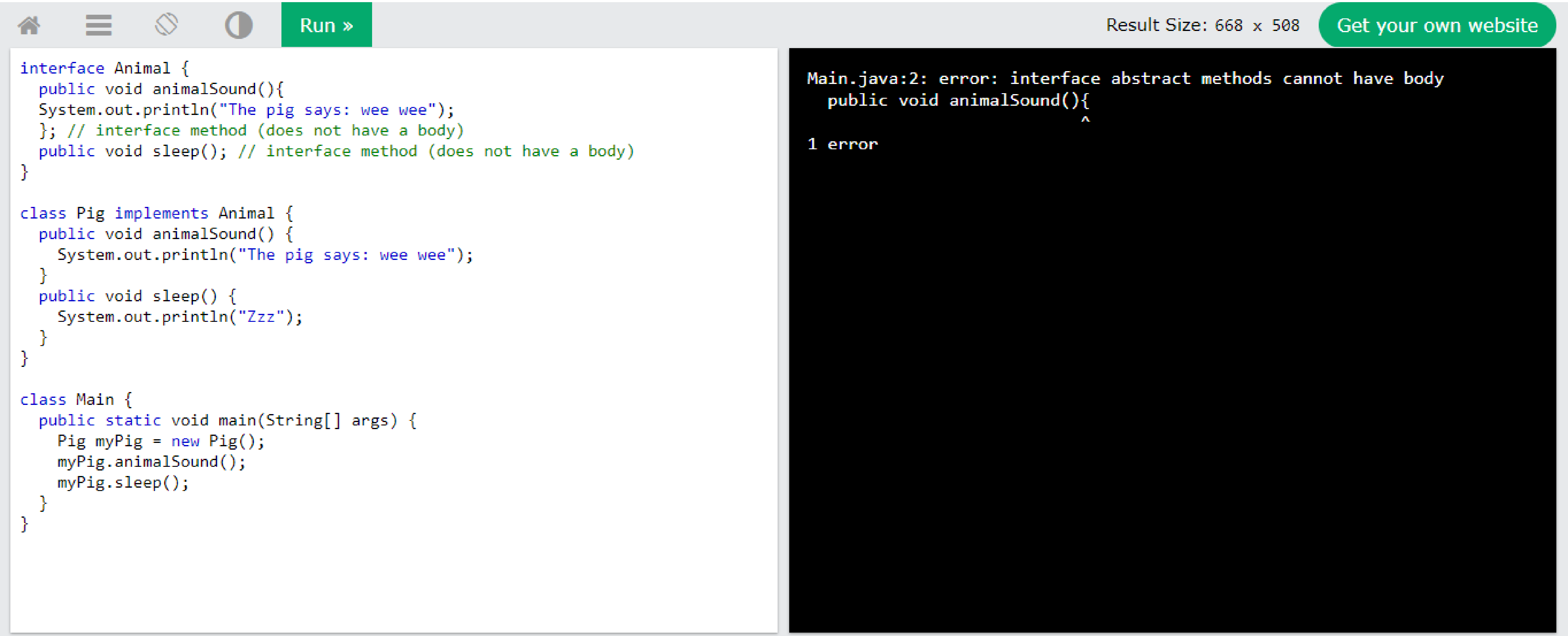
Incorrect example using Interface



```java
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();
    myPig.animalSound();
    myPig.sleep();
  }
}
```

```
Main.java:6: error: no interface expected here
class Pig extends Animal {
                  ^
1 error
```

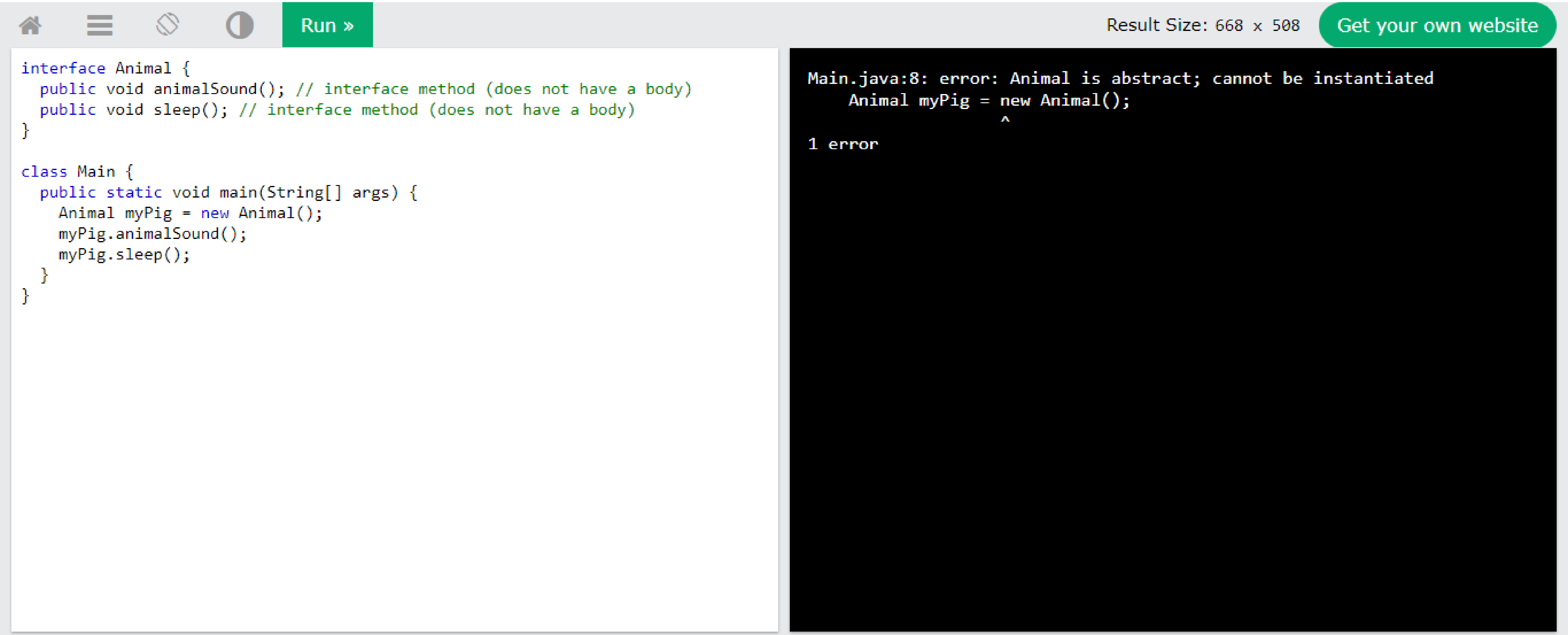
Incorrect example using Interface



```java
interface Animal {
  public void animalSound(){
  System.out.println("The pig says: wee wee");
  }; // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

class Pig implements Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();
    myPig.animalSound();
    myPig.sleep();
  }
}
```

```
Main.java:2: error: interface abstract methods cannot have body
  public void animalSound(){
                           ^
1 error
```

Incorrect example using Interface



```java
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

class Main {
  public static void main(String[] args) {
    Animal myPig = new Animal();
    myPig.animalSound();
    myPig.sleep();
  }
}
```

```
Main.java:8: error: Animal is abstract; cannot be instantiated
    Animal myPig = new Animal();
                   ^
1 error
```

When to use interfaces, and when to use abstract classes?

The abstract class is used to implement the Template Method pattern. While the interface is used (among other things) to implement the Observer pattern. Use interfaces if one class wants to have multiple data types.