AP Computer Science A

UNIT 2 TOPIC 2
Modifying Object State

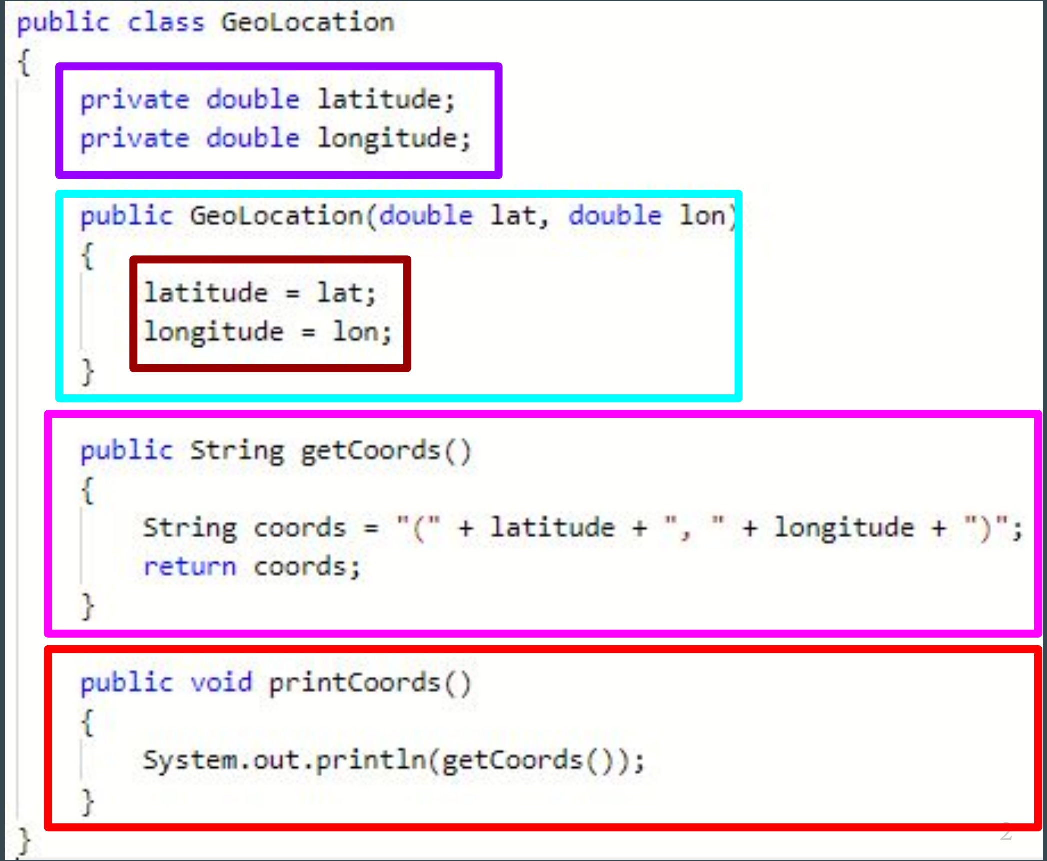
1. Work with your partner on the Vocab Warm Up

Vocab Review

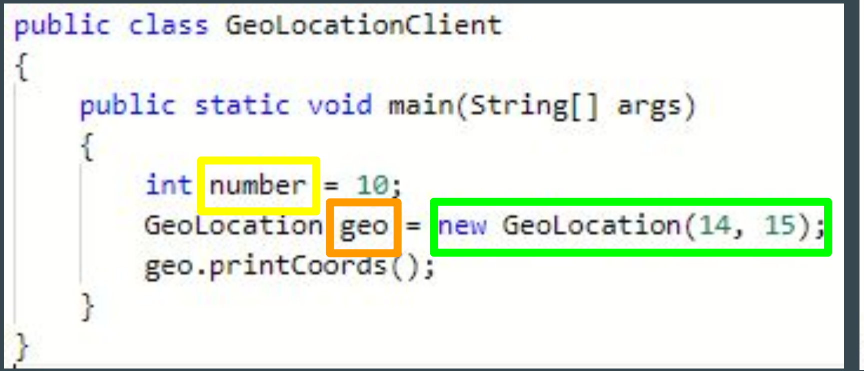
Method that returns a value
Method that does not return a value
Constructor
Overloaded

MATCH each term/phrase with the appropriate color; *one term has no match!*

Attributes
Properties
Fields
Instance Variables
Saved data ("state") of an object
Primitive variable
Reference variable
Creating an object
Creating an instance of a class
Initializing instance variables



```java
public class GeoLocation
{
    private double latitude;
    private double longitude;

    public GeoLocation(double lat, double lon)
    {
        latitude = lat;
        longitude = lon;
    }

    public String getCoords()
    {
        String coords = "(" + latitude + ", " + longitude + ")";
        return coords;
    }

    public void printCoords()
    {
        System.out.println(getCoords());
    }
}
```



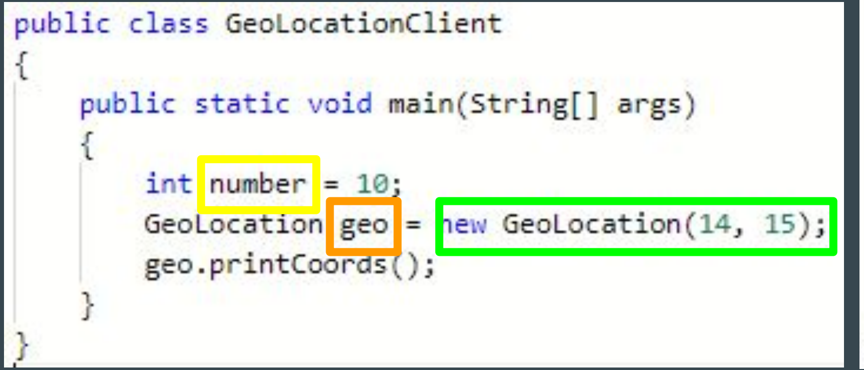
```java
public class GeoLocationClient
{
    public static void main(String[] args)
    {
        int number = 10;
        GeoLocation geo = new GeoLocation(14, 15);
        geo.printCoords();
    }
}
```

Vocab Review

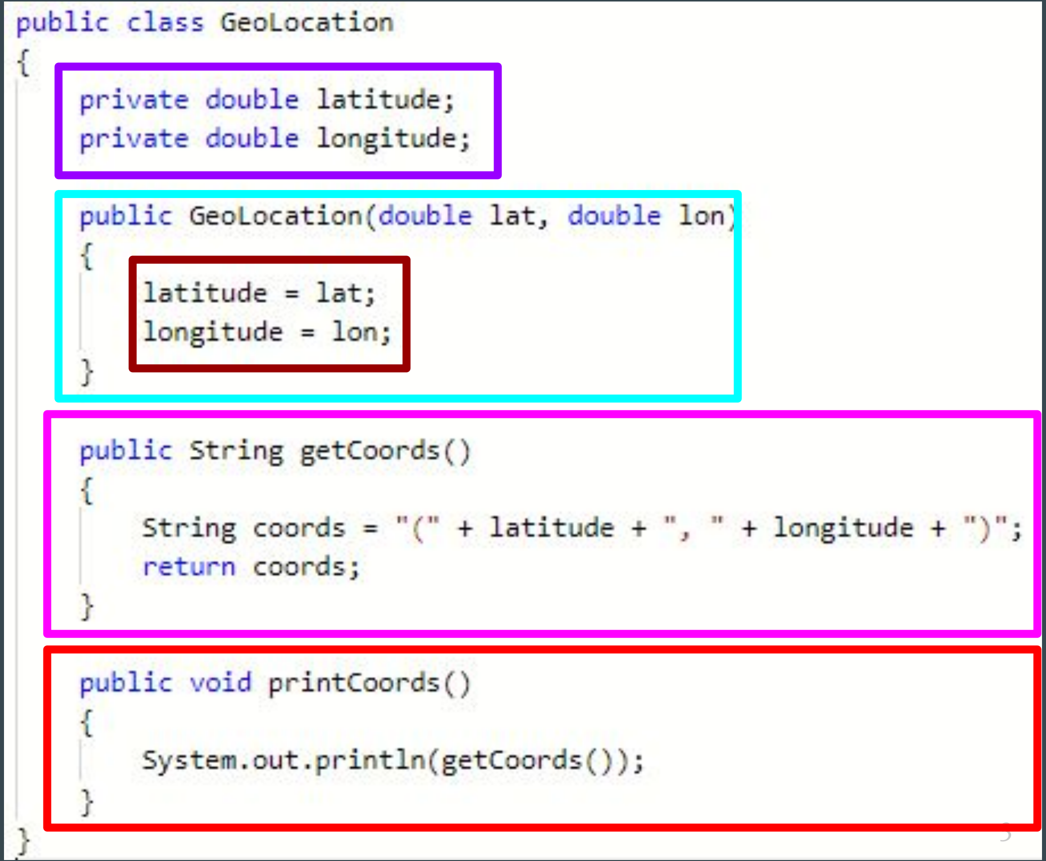
Method that returns a value
Method that does not return a value
Constructor
~~Overloaded~~
no match!

MATCH each term/phrase with the appropriate color; *one term has no match!*

Attributes
Properties
Fields
Instance Variables
Saved data ("state") of an object
Primitive variable
Reference variable
Creating an object
Creating an instance of a class
Initializing instance variables



```
public class GeoLocationClient
{
    public static void main(String[] args)
    {
        int number = 10;
        GeoLocation geo = new GeoLocation(14, 15);
        geo.printCoords();
    }
}
```

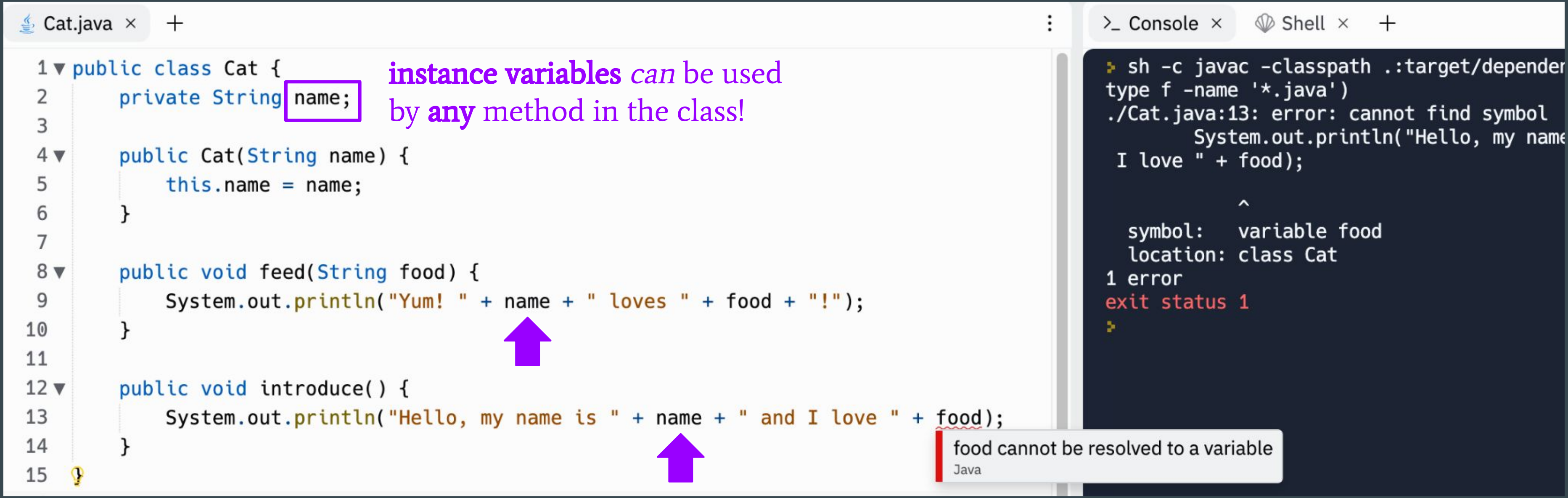


```
public class GeoLocation
{
    private double latitude;
    private double longitude;

    public GeoLocation(double lat, double lon)
    {
        latitude = lat;
        longitude = lon;
    }

    public String getCoords()
    {
        String coords = "(" + latitude + ", " + longitude + ")";
        return coords;
    }

    public void printCoords()
    {
        System.out.println(getCoords());
    }
}
```

Partner Coding Warm Up

Open the U2T2 Lab 2 Partner Challenges and work on the Partner Warm Up problem; we will come back to discuss in 10 minutes.

Instance Variable vs. Parameters



Cat.java

```java
public class Cat {
    private String name;

    public Cat(String name) {
        this.name = name;
    }

    public void feed(String food) {
        System.out.println("Yum! " + name + " loves " + food + "!");
    }

    public void introduce() {
        System.out.println("Hello, my name is " + name + " and I love " + food);
    }
}
```

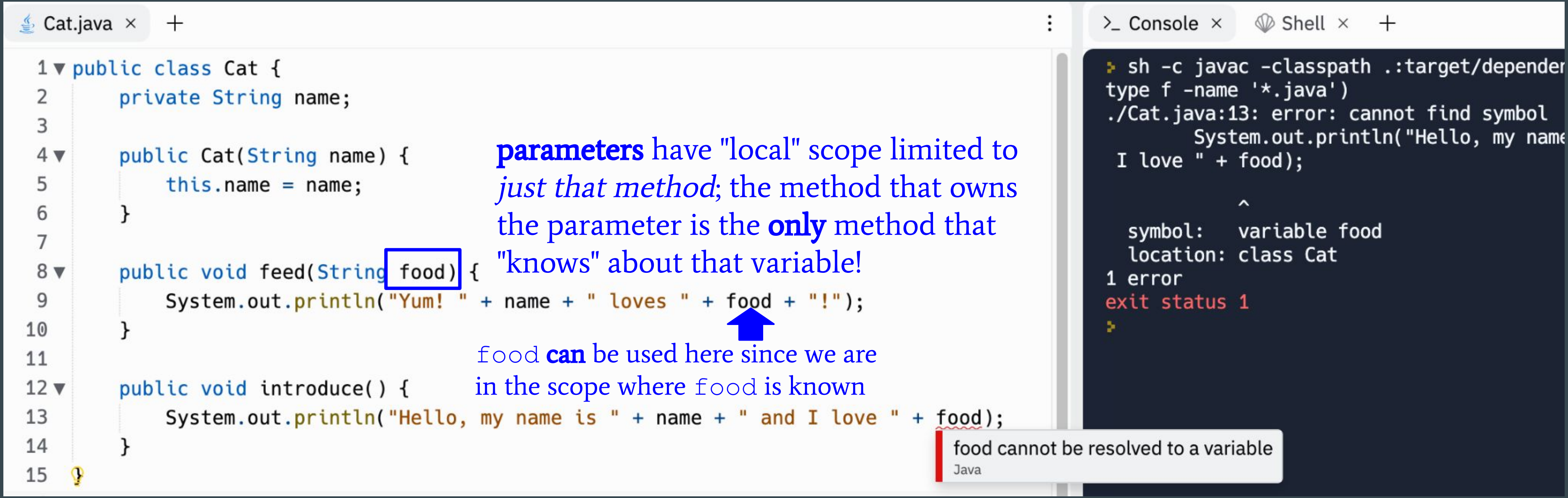
instance variables *can* be used by **any** method in the class!

food cannot be resolved to a variable
Java

Console

```
sh -c javac -classpath .:target/depender
type f -name '*.java')
./Cat.java:13: error: cannot find symbol
        System.out.println("Hello, my name
 I love " + food);

                    ^
  symbol:   variable food
  location: class Cat
1 error
exit status 1
```

Instance Variable vs. Parameters



Cat.java

```java
public class Cat {
    private String name;

    public Cat(String name) {
        this.name = name;
    }

    public void feed(String food) {
        System.out.println("Yum! " + name + " loves " + food + "!");
    }

    public void introduce() {
        System.out.println("Hello, my name is " + name + " and I love " + food);
    }
}
```

parameters have "local" scope limited to *just that method*; the method that owns the parameter is the **only** method that "knows" about that variable!

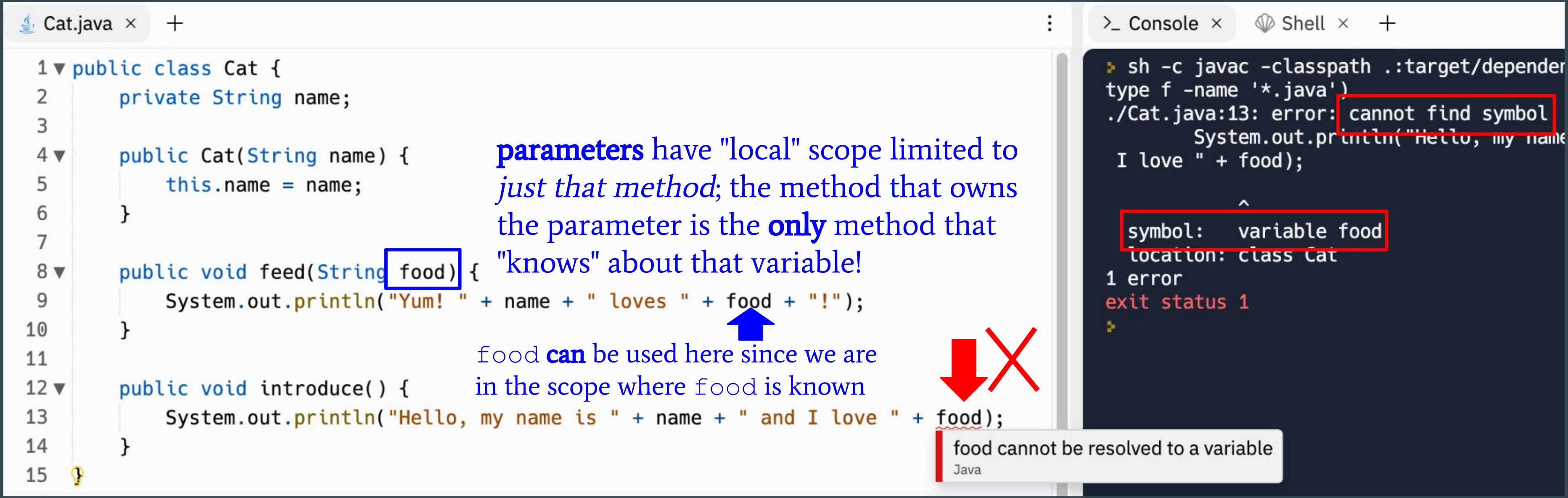
`food` **can** be used here since we are in the scope where `food` is known

food cannot be resolved to a variable
Java

Console

```
> sh -c javac -classpath .:target/depender
type f -name '*.java')
./Cat.java:13: error: cannot find symbol
                System.out.println("Hello, my name
 I love " + food);

                        ^
  symbol:   variable food
  location: class Cat
1 error
exit status 1
>
```

Instance Variable vs. Parameters



```
public class Cat {
    private String name;

    public Cat(String name) {
        this.name = name;
    }

    public void feed(String food) {
        System.out.println("Yum! " + name + " loves " + food + "!");
    }

    public void introduce() {
        System.out.println("Hello, my name is " + name + " and I love " + food);
    }
}
```

food cannot be resolved to a variable
Java

```
> sh -c javac -classpath .:target/depender
type f -name '*.java')
./Cat.java:13: error: cannot find symbol
        System.out.println("Hello, my name
 I love " + food);

                         ^
  symbol:   variable food
  location: class Cat
1 error
exit status 1
>
```

parameters have "local" scope limited to *just that method*; the method that owns the parameter is the **only** method that "knows" about that variable!

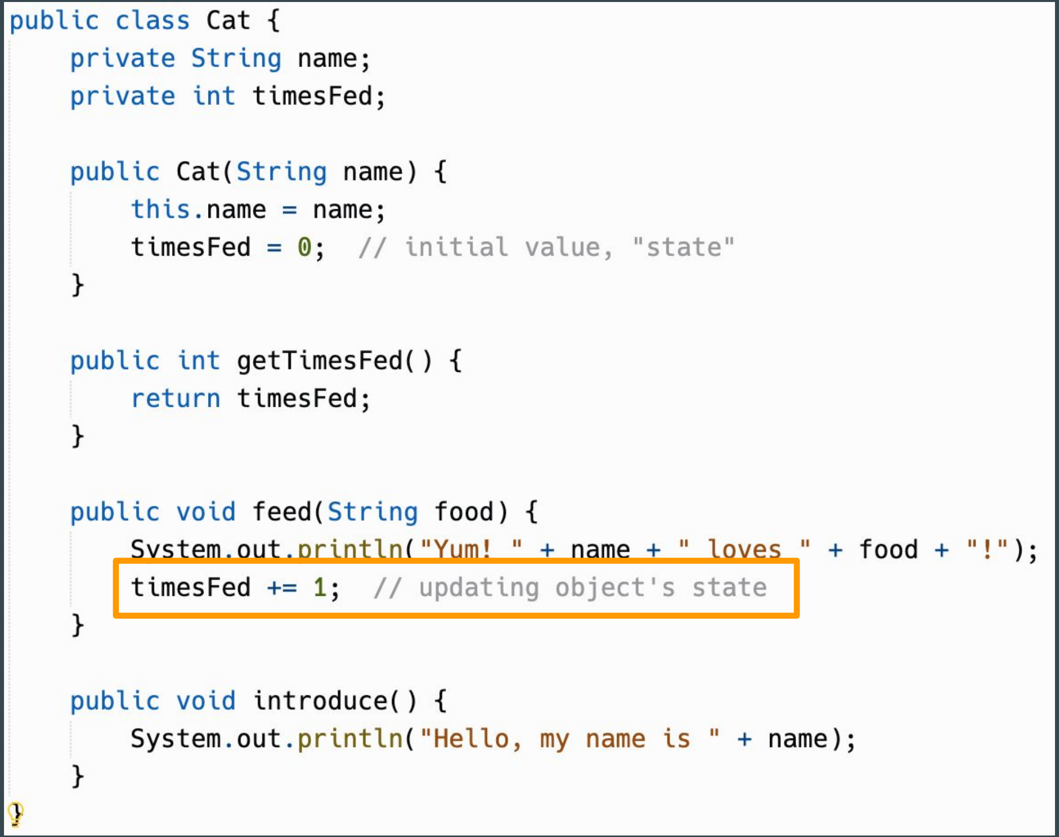
`food` **can** be used here since we are in the scope where `food` is known

can't use `food` here, since `food` is only known by the `feed` method; the `introduce` method doesn't have access to another method's "scope" or its parameters

Modifying Object State

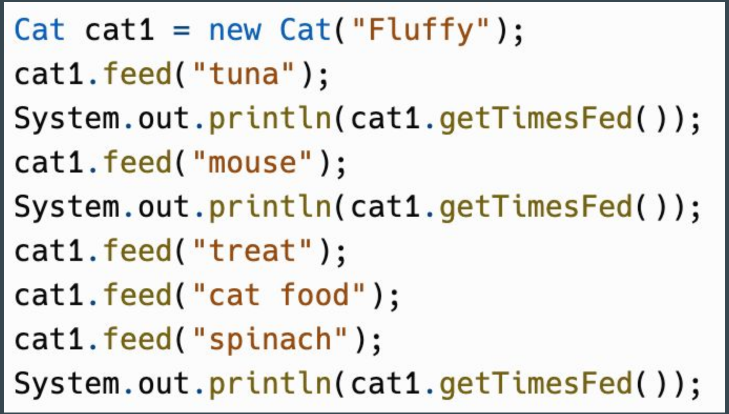
The **state** of an object just means the value of its instance variables at any given point during a program's execution.

The **state** of an object can be modified using setter methods **or** by other methods of the class; in the example on the right, the `feed` method modifies the `timesFed` instance variable, thus modifying the state of the object

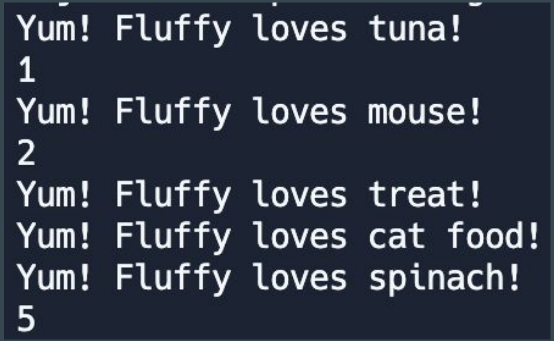


```java
public class Cat {
    private String name;
    private int timesFed;

    public Cat(String name) {
        this.name = name;
        timesFed = 0;   // initial value, "state"
    }

    public int getTimesFed() {
        return timesFed;
    }

    public void feed(String food) {
        System.out.println("Yum! " + name + " loves " + food + "!");
        timesFed += 1;   // updating object's state
    }

    public void introduce() {
        System.out.println("Hello, my name is " + name);
    }
```

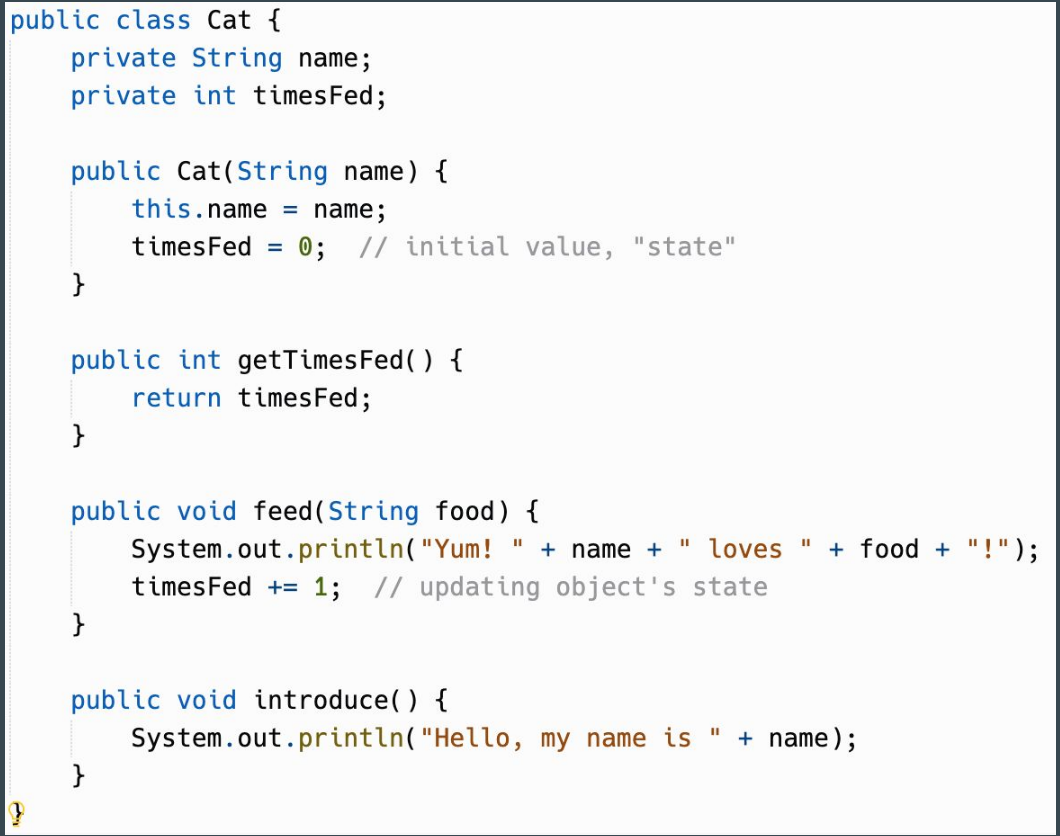
Modifying Object State



```java
Cat cat1 = new Cat("Fluffy");
cat1.feed("tuna");
System.out.println(cat1.getTimesFed());
cat1.feed("mouse");
System.out.println(cat1.getTimesFed());
cat1.feed("treat");
cat1.feed("cat food");
cat1.feed("spinach");
System.out.println(cat1.getTimesFed());
```



```
Yum! Fluffy loves tuna!
1
Yum! Fluffy loves mouse!
2
Yum! Fluffy loves treat!
Yum! Fluffy loves cat food!
Yum! Fluffy loves spinach!
5
```



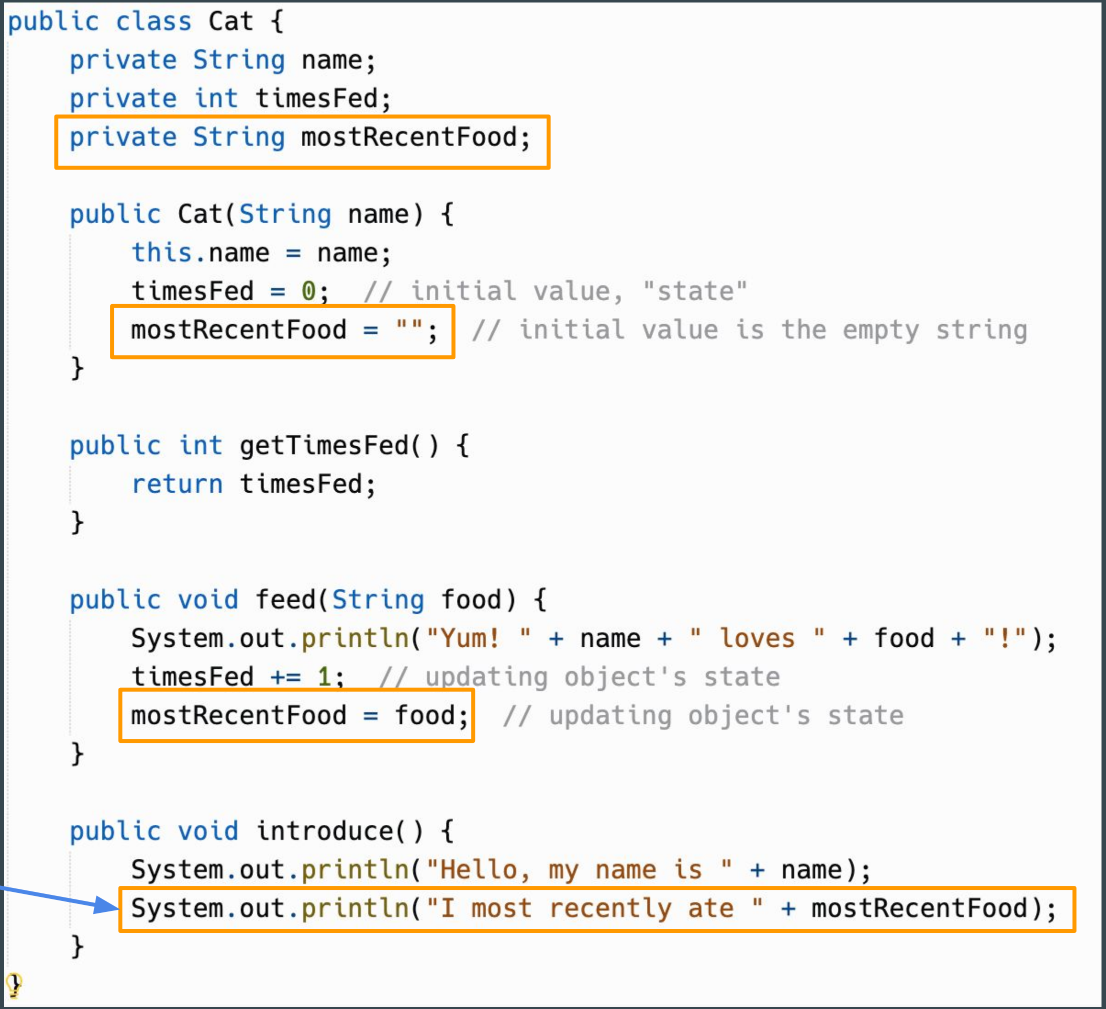
```java
public class Cat {
    private String name;
    private int timesFed;

    public Cat(String name) {
        this.name = name;
        timesFed = 0;  // initial value, "state"
    }

    public int getTimesFed() {
        return timesFed;
    }

    public void feed(String food) {
        System.out.println("Yum! " + name + " loves " + food + "!");
        timesFed += 1;  // updating object's state
    }

    public void introduce() {
        System.out.println("Hello, my name is " + name);
    }
```

The **state** of the object is changing as the program runs because it's `timesFed` instance variable gets updated each time `feed` is called!

Modifying Object State

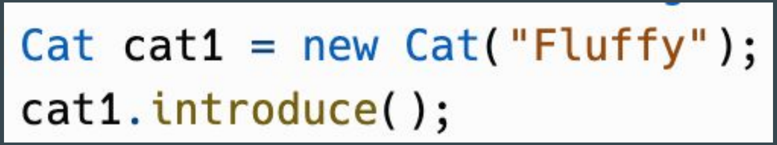
In the example on the right, the `feed` method is updated to also modifies a new instance variable, `mostRecentFood`, which we added so that we could track another changing value

We can access the `mostRecentFood` instance variable from *any* method!

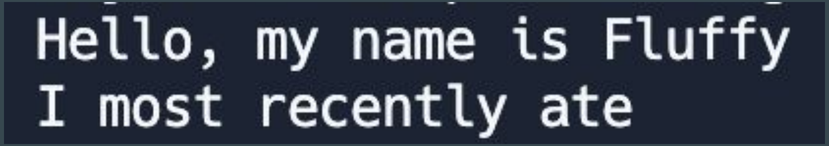


```java
public class Cat {
    private String name;
    private int timesFed;
    private String mostRecentFood;

    public Cat(String name) {
        this.name = name;
        timesFed = 0;  // initial value, "state"
        mostRecentFood = "";  // initial value is the empty string
    }

    public int getTimesFed() {
        return timesFed;
    }

    public void feed(String food) {
        System.out.println("Yum! " + name + " loves " + food + "!");
        timesFed += 1;  // updating object's state
        mostRecentFood = food;  // updating object's state
    }

    public void introduce() {
        System.out.println("Hello, my name is " + name);
        System.out.println("I most recently ate " + mostRecentFood);
    }
}
```

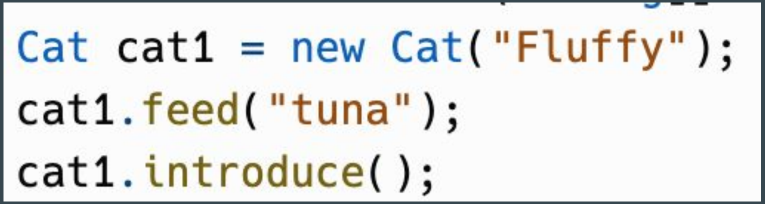
Modifying Object State



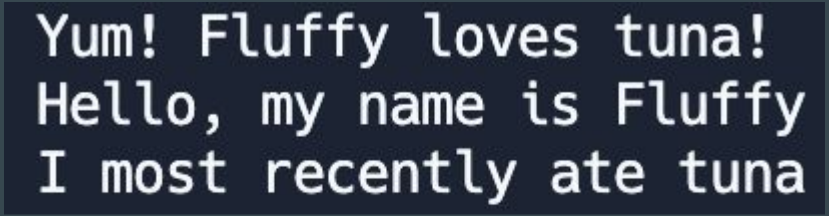
```java
Cat cat1 = new Cat("Fluffy");
cat1.introduce();
```



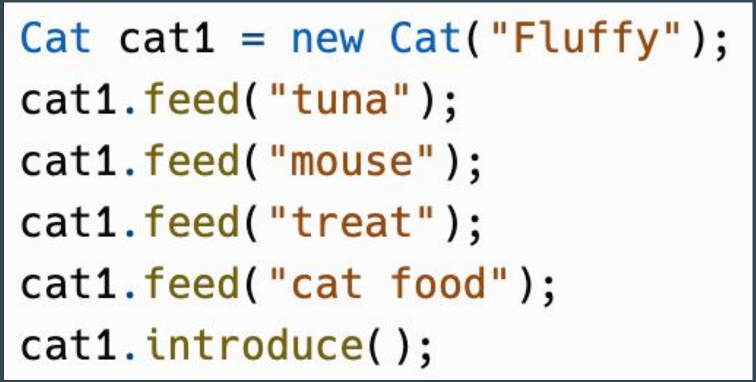
```
Hello, my name is Fluffy
I most recently ate
```



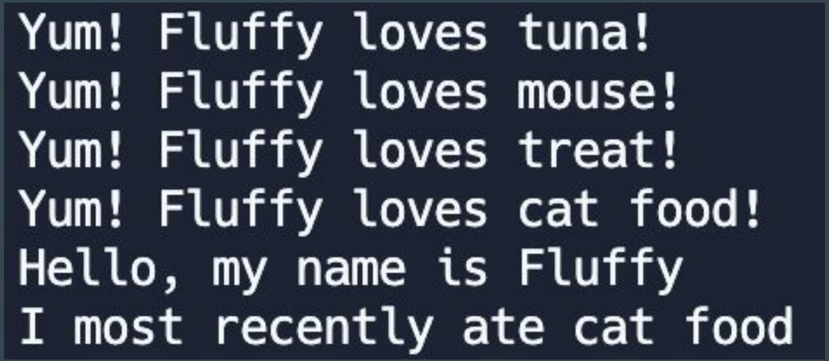
```java
Cat cat1 = new Cat("Fluffy");
cat1.feed("tuna");
cat1.introduce();
```



```
Yum! Fluffy loves tuna!
Hello, my name is Fluffy
I most recently ate tuna
```



```java
Cat cat1 = new Cat("Fluffy");
cat1.feed("tuna");
cat1.feed("mouse");
cat1.feed("treat");
cat1.feed("cat food");
cat1.introduce();
```

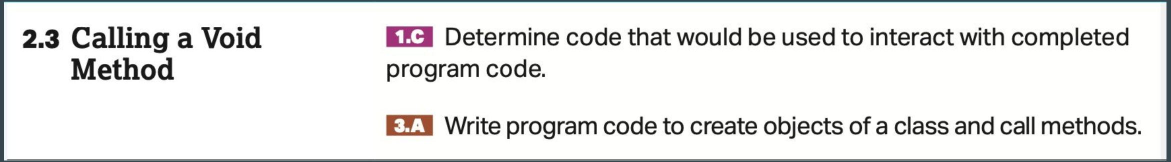


```
Yum! Fluffy loves tuna!
Yum! Fluffy loves mouse!
Yum! Fluffy loves treat!
Yum! Fluffy loves cat food!
Hello, my name is Fluffy
I most recently ate cat food
```

UNIT 2 Using Objects

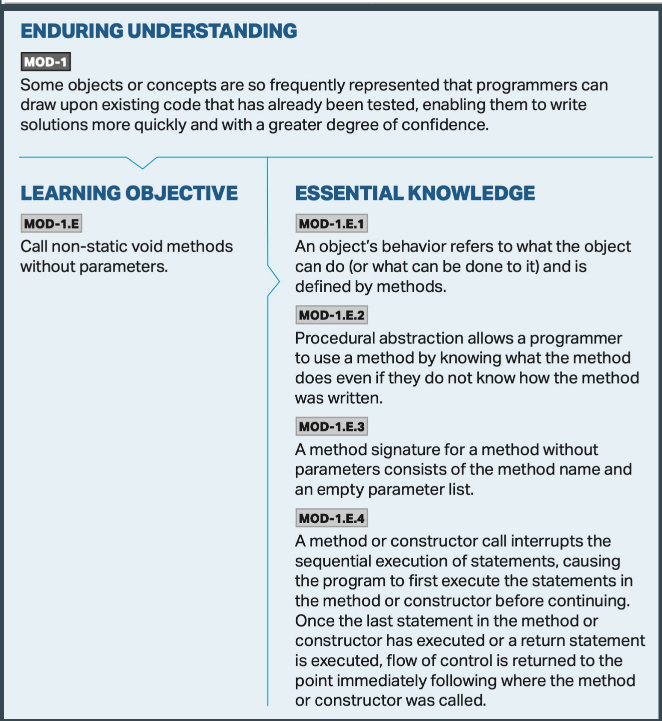
CollegeBoard Standards

Unit 2 Topic 3


2.3 Calling a Void Method

1.C Determine code that would be used to interact with completed program code.

3.A Write program code to create objects of a class and call methods.


ENDURING UNDERSTANDING

MOD-1
Some objects or concepts are so frequently represented that programmers can draw upon existing code that has already been tested, enabling them to write solutions more quickly and with a greater degree of confidence.

LEARNING OBJECTIVE

MOD-1.E
Call non-static void methods without parameters.

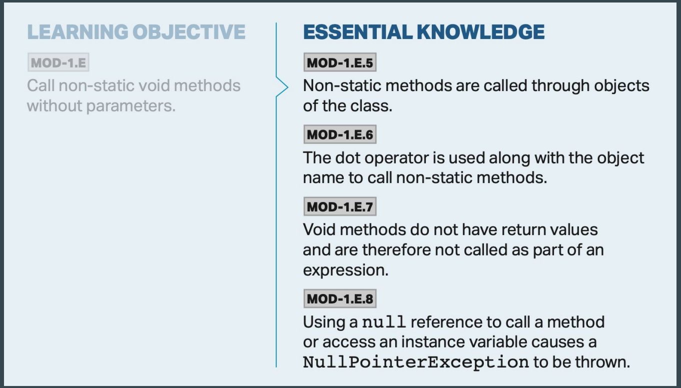
ESSENTIAL KNOWLEDGE

MOD-1.E.1
An object's behavior refers to what the object can do (or what can be done to it) and is defined by methods.

MOD-1.E.2
Procedural abstraction allows a programmer to use a method by knowing what the method does even if they do not know how the method was written.

MOD-1.E.3
A method signature for a method without parameters consists of the method name and an empty parameter list.

MOD-1.E.4
A method or constructor call interrupts the sequential execution of statements, causing the program to first execute the statements in the method or constructor before continuing. Once the last statement in the method or constructor has executed or a return statement is executed, flow of control is returned to the point immediately following where the method or constructor was called.


LEARNING OBJECTIVE

MOD-1.E
Call non-static void methods without parameters.

ESSENTIAL KNOWLEDGE

MOD-1.E.5
Non-static methods are called through objects of the class.

MOD-1.E.6
The dot operator is used along with the object name to call non-static methods.

MOD-1.E.7
Void methods do not have return values and are therefore not called as part of an expression.

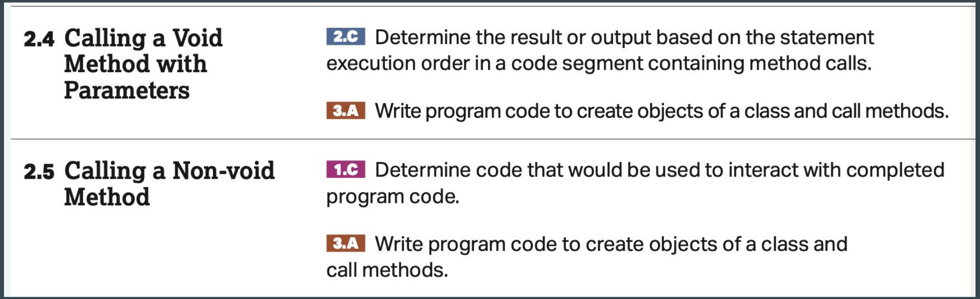
MOD-1.E.8
Using a `null` reference to call a method or access an instance variable causes a `NullPointerException` to be thrown.

UNIT 2

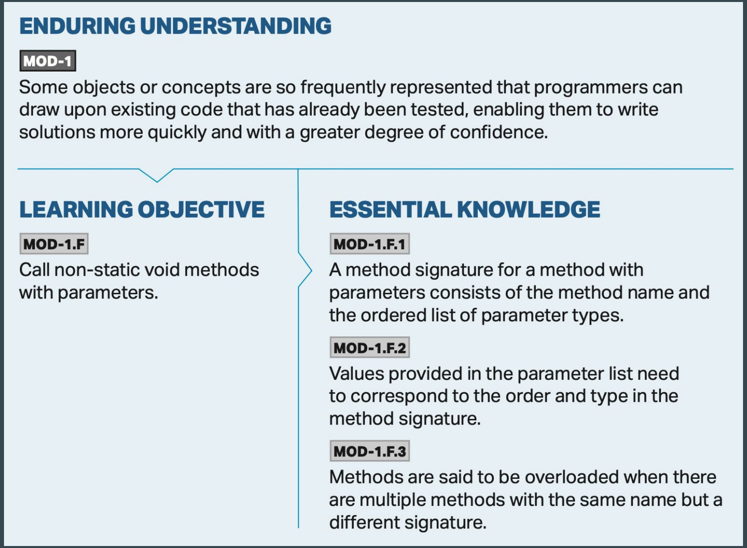
Using Objects

CollegeBoard Standards

Unit 2 Topics 4 & 5



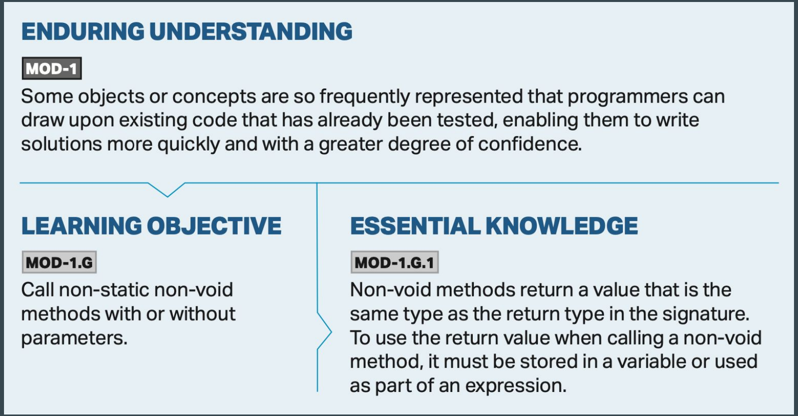
2.4 Calling a Void Method with Parameters	**2.C** Determine the result or output based on the statement execution order in a code segment containing method calls. **3.A** Write program code to create objects of a class and call methods.
2.5 Calling a Non-void Method	**1.C** Determine code that would be used to interact with completed program code. **3.A** Write program code to create objects of a class and call methods.



ENDURING UNDERSTANDING

MOD-1
Some objects or concepts are so frequently represented that programmers can draw upon existing code that has already been tested, enabling them to write solutions more quickly and with a greater degree of confidence.

LEARNING OBJECTIVE	ESSENTIAL KNOWLEDGE
MOD-1.F Call non-static void methods with parameters.	**MOD-1.F.1** A method signature for a method with parameters consists of the method name and the ordered list of parameter types. **MOD-1.F.2** Values provided in the parameter list need to correspond to the order and type in the method signature. **MOD-1.F.3** Methods are said to be overloaded when there are multiple methods with the same name but a different signature.



ENDURING UNDERSTANDING

MOD-1
Some objects or concepts are so frequently represented that programmers can draw upon existing code that has already been tested, enabling them to write solutions more quickly and with a greater degree of confidence.

LEARNING OBJECTIVE	ESSENTIAL KNOWLEDGE
MOD-1.G Call non-static non-void methods with or without parameters.	**MOD-1.G.1** Non-void methods return a value that is the same type as the return type in the signature. To use the return value when calling a non-void method, it must be stored in a variable or used as part of an expression.

Agenda

- U2T2 Lab 2: Partner Programming Challenges (due **tomorrow**)

TOMORROW!

We will do some mixed partner AP review, followed by a MCQ quiz at the end of the period in AP Classroom.