AP Computer Science A

UNIT 2 TOPICS 1 & 2
Introduction to Objects

Class 01

Do Now!

InputDemo.java

```
1   import java.util.Scanner;
2
3   public class InputDemo
4   {
5       public static void main(String[] args)
6       {
7           Scanner scan = new Scanner(System.in);
8
9           System.out.print("Please enter your name: ");
10          String name = scan.nextLine();
11
12          System.out.print("How many apples? ");
13          int apples = scan.nextInt();
14
15          System.out.print("Price per apple? ");
16          double cost = scan.nextDouble();
17
18          System.out.println(name + " has " + apples + " apples that cost " + cost + " each.");
19      }
20  }
```

Identify ALL the different classes in this program. What are some similarities and/or differences that you notice?

UNIT 2 Using Objects

College Board Standards

Unit 2 Topic 1

MOD-1

2.1 Objects: Instances of Classes

5.A Describe the behavior of a given segment of program code.

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.B

Explain the relationship between a class and an object.

ESSENTIAL KNOWLEDGE

MOD-1.B.1

An object is a specific instance of a class with defined attributes.

MOD-1.B.2

A class is the formal implementation, or blueprint, of the attributes and behaviors of an object.

UNIT 2 Using Objects

College Board Standards
Unit 2 Topic 2

MOD-1 VAR-1

2.2 Creating and Storing Objects (Instantiation)

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.C

Identify, using its signature, the correct constructor being called.

ESSENTIAL KNOWLEDGE

MOD-1.C.1

A signature consists of the constructor name and the parameter list.

MOD-1.C.2

The parameter list, in the header of a constructor, lists the types of the values that are passed and their variable names. These are often referred to as formal parameters.

MOD-1.C.3

A parameter is a value that is passed into a constructor. These are often referred to as actual parameters.

MOD-1.C.4

Constructors are said to be overloaded when there are multiple constructors with the same name but a different signature.

MOD-1.C.5

The actual parameters passed to a constructor must be compatible with the types identified in the formal parameter list.

MOD-1.C.6

Parameters are passed using call by value. Call by value initializes the formal parameters with copies of the actual parameters.

LEARNING OBJECTIVE

MOD-1.D

For creating objects:

a. Create objects by calling constructors without parameters.

b. Create objects by calling constructors with parameters.

ESSENTIAL KNOWLEDGE

MOD-1.D.1

Every object is created using the keyword `new` followed by a call to one of the class's constructors.

MOD-1.D.2

A class contains constructors that are invoked to create objects. They have the same name as the class.

MOD-1.D.3

Existing classes and class libraries can be utilized as appropriate to create objects.

MOD-1.D.4

Parameters allow values to be passed to the constructor to establish the initial state of the object.

ENDURING UNDERSTANDING

VAR-1

To find specific solutions to generalizable problems, programmers include variables in their code so that the same algorithm runs using different input values.

LEARNING OBJECTIVE

VAR-1.D

Define variables of the correct types to represent reference data.

ESSENTIAL KNOWLEDGE

VAR-1.D.1

The keyword `null` is a special value used to indicate that a reference is not associated with any object.

VAR-1.D.2

The memory associated with a variable of a reference type holds an object reference value or, if there is no object, `null`. This value is the memory address of the referenced object.

Do Now!

InputDemo.java

```
1   import java.util.Scanner;
2
3   public class InputDemo
4   {
5       public static void main(String[] args)
6       {
7           Scanner scan = new Scanner(System.in);
8
9           System.out.print("Please enter your name: ");
10          String name = scan.nextLine();
11
12          System.out.print("How many apples? ");
13          int apples = scan.nextInt();
14
15          System.out.print("Price per apple? ");
16          double cost = scan.nextDouble();
17
18          System.out.println(name + " has " + apples + " apples that cost " + cost + " each.");
19      }
20  }
```

Identify ALL the different classes in this program. What are some similarities and/or differences that you notice?

String
Scanner
System
InputDemo

Anything with a Capital letter is a class!

Do Now!

InputDemo.java

```
1   import java.util.Scanner;
2
3   public class InputDemo
4   {
5       public static void main(String[] args)
6       {
7           Scanner scan = new Scanner(System.in);
8
9           System.out.print("Please enter your name: ");
10          String name = scan.nextLine();
11
12          System.out.print("How many apples? ");
13          int apples = scan.nextInt();
14
15          System.out.print("Price per apple? ");
16          double cost = scan.nextDouble();
17
18          System.out.println(name + " has " + apples + " apples that cost " + cost + " each.");
19      }
20  }
```

Identify ALL the different classes in this program. What are some similarities and/or differences that you notice?

String
Scanner
System
InputDemo

Anything with a Capital letter is a class!

String, System, Scanner are Java classes that we USE in our *custom* class, InputDemo

Do Now!

InputDemo.java

```
1   import java.util.Scanner;
2
3   public class InputDemo
4   {
5       public static void main(String[] args)
6       {
7           Scanner scan = new Scanner(System.in);
8
9           System.out.print("Please enter your name: ");
10          String name = scan.nextLine();
11
12          System.out.print("How many apples? ");
13          int apples = scan.nextInt();
14
15          System.out.print("Price per apple? ");
16          double cost = scan.nextDouble();
17
18          System.out.println(name + " has " + apples + " apples that cost " + cost + " each.");
19      }
20  }
```

Identify ALL the different classes in this program. What are some similarities and/or differences that you notice?

String
Scanner
System
InputDemo

Anything with a Capital letter is a class!

String, System, Scanner are Java classes that we USE in our *custom* class, InputDemo

We see the code for the InputDemo class (here it is!) but not the other classes

Do Now!

InputDemo.java

```
1   import java.util.Scanner;
2
3   public class InputDemo
4   {
5       public static void main(String[] args)
6       {
7           Scanner scan = new Scanner(System.in);
8
9           System.out.print("Please enter your name: ");
10          String name = scan.nextLine();
11
12          System.out.print("How many apples? ");
13          int apples = scan.nextInt();
14
15          System.out.print("Price per apple? ");
16          double cost = scan.nextDouble();
17
18          System.out.println(name + " has " + apples + " apples that cost " + cost + " each.");
19      }
20  }
```

Identify ALL the different classes in this program. What are some similarities and/or differences that you notice?

String
Scanner
System
InputDemo

Anything with a Capital letter is a class!

String, System, Scanner are Java classes that we USE in our *custom* class, InputDemo

We see the code for the InputDemo class (here it is!) but not the other classes

We must import Scanner, but no other classes

"Client" class

- A class that *uses* another class in its own code is considered a "**client**" of the class that it uses
- Therefore, we can say that `InputDemo` is a "**client**" of the `Scanner`, `String`, and `System` classes, since it is *using* those classes as part of its own code

Agenda

- **Intro to objects!**
- U2T1 Lab

Where we are heading!

- Up until now, we have been using classes that *other people have written* to write our own programs.

Where we are heading!

- Up until now, we have been using classes that *other people have written* to write our own programs.
- For example, we have created **objects** of type **`String`** using the **`String`** class and **objects** of type **`Scanner`** using the **`Scanner`** class:

Where we are heading!

- Up until now, we have been using classes that *other people have written* to write our own programs.
- For example, we have created **objects** of type **`String`** using the **`String`** class and **objects** of type **`Scanner`** using the **`Scanner`** class:

```
Scanner scan = new Scanner(System.in);
String name = "Mr. Miller";
```

`scan` is an **object** of type `Scanner`, and `name` is an **object** of type `String`

Where we are heading!

- Up until now, we have been using classes that *other people have written* to write our own programs.
- For example, we have created **objects** of type **`String`** using the **`String`** class and **objects** of type **`Scanner`** using the **`Scanner`** class:

```
Scanner scan = new Scanner(System.in);
String name = "Mr. Miller";
```

`scan` is an **object** of type `Scanner`, and `name` is an **object** of type `String`

- In the project, you will be creating an **object** of type **`DecimalFormatter`**:

```
DecimalFormatter formatter = new DecimalFormatter("#.##");
```

`formatter` is an **object** of type `DecimalFormatter`

Where we are heading!

- Up until now, we have been using classes that *other people have written* to write our own programs.
- For example, we have created **objects** of type **`String`** using the **`String`** class and **objects** of type **`Scanner`** using the **`Scanner`** class:

```
Scanner scan = new Scanner(System.in);
String name = "Mr. Miller";
```

```
int num = 5;
```

`num` is **NOT** an **object,** WHY NOT?

Where we are heading!

- Up until now, we have been using classes that *other people have written* to write our own programs.
- For example, we have created **objects** of type **`String`** using the **`String`** class and **objects** of type **`Scanner`** using the **`Scanner`** class:

```
Scanner scan = new Scanner(System.in);
String name = "Mr. Miller";
```

```
int num = 5;
```

`num` is **NOT** an **object**, it is a **primitive**! Only from *classes* can we create *objects*.

Where we are heading!

- We have also used **called methods on objects**:

```
System.out.println(name);    // calling the println method

scan.nextLine();             // calling the nextLine method
```

Where we are heading!

- We have also used **called methods** on objects:

```
System.out.println(name);    // calling the println method

scan.nextLine();             // calling the nextLine method
```

- When we use Scanner, System, and String classes, and call methods on objects of those classes, we are using code that *other people have written* (in this case, the Java people, as part of the Java language libraries).

Where we are heading!

- But the fun part of Java programming is creating *new* classes that you come up with to represent your *own* ideas and build your own programs using your own classes!

Classes & Objects: Overview

- A **class definition** (what you type in the .java file) is used to define a new data type (a new class) by defining the **attributes** and **behaviors** of the objects that are created from that class

Partner Activity!

- Think of anything that you can *model* in a computer program (a Student, a Cat, a Teacher, Building, Restaurant, Video Game, anything you can think of)
- For the thing that you chose, think of three **attributes**. Use a combination of integer, double, String, and boolean.
- Then think of one **behavior** of that thing.

EXAMPLE:

My class: `Cat`

attributes

```
String name
int age
double weight
```

behavior

```
catchMouse
```

Classes & Objects: Overview

- **Objects** *are created from the* **class definition** and are specific **instances** of a class.
- Think of the **class definition** as a “blueprint” or “cookie cutter” for creating different **objects** of that class type; each object made from the blueprint is its own "instance of the class" from which it was created!

Creating Objects

- You can make as many objects as you want from a class!
- Each object (instance) that you create is unique and has its own set of “attributes”, in other words, its own set of data (values) that are saved with that object/instance. All objects can perform behava

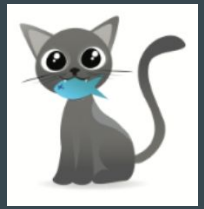
Here are three Cat objects created from a Cat class (blueprint):

Each of these is a unique "instance" of the Cat class

cat1

name = “Bubbles”
age = 2
weight = 3.5

cat2



name = “George”
age = 5
weight = 4.7

cat3

name = “Monster”
age = 7
weight = 6.2

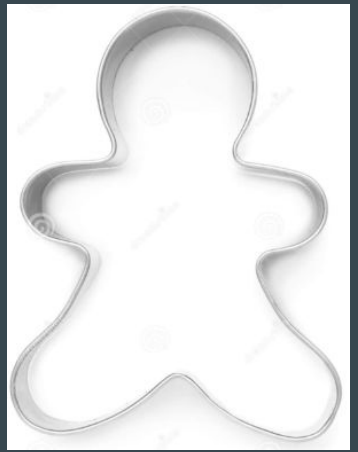
each Cat object has its own *unique* set of the three *attributes* "stored", and *all* can perform the catch mouse behavior!

Classes & Objects: Overview

As an analogy, think of each **GingerBreadMan cookie** as a **GingerBreadMan** object, or "instance of" the **GingerBreadMan cookie cutter** ("blueprint" or class) from which is was pressed; while they all come from the same "blueprint" (class), they are each unique with their own unique attributes!

GingerBreadMan objects, or "instances" of the GingerBreadMan class (cookie cutter)

This represents the GingerBreadMan class, or the cookie cutter ("blueprint") that ***produces*** GingerBreadMan cookies (objects, or instances)



Objects & Classes

DEMO: Rectangles

Classes & Objects: Overview

- Special methods called **constructors** are used to create objects; here are two examples of using constructors to create objects (note the `new` keyword!):

```
Scanner scan = new Scanner(System.in);
DecimalFormat formatter = new DecimalFormat("#.##");
```

"Runner" and "Client" classes

- In our demo, the `Rectangle` class is a "blueprint" class used to create objects -- it is not an "executable" class, and has no `main` method.
- Our `RectangleRunner` class *is* an "executable" class since it has a `main` method, and we often refer to a class with a `main` methods as a "**runner**" class
  - `RectangleRunner` is a "**runner**" class in this demo
- Additionally, our `RectangleRunner` class *uses* the `Rectangle` class as a "blueprint" to create and use `Rectangle` **objects** (or "**instances**" of `Rectangle`), and so we say that `RectangleRunner` is a "**client**" of `Rectangle`

Agenda

- U2L1
- Finish early? Work on your project or explore a bit more!

Before you leave...

- Make sure you have shared your project to GitHub (Git → GitHub → Share Project on GitHub), and Git → Pushed your most recent code!
- Log out of your GitHub account on IntelliJ (File → Settings → Version Control → GitHub → click "-" by your name
- Open up a *NON*-incognito Chrome window, go to github.com, and make sure you are logged out there
- Close your project in IntelliJ (File → Close Project) and remove it from "Recents" (use the gear icon)

Summary

- A **class** is the formal implementation, or blueprint, of the **attributes** (**instance variables**) and **behaviors** (**methods**) of an object; an **object** is a specific **instance** of a class.
- A class contains **constructors** that are called to create objects and have the same name as the class.
- A **constructor signature** consists of the constructor name and the parameter list.
- Constructors are said to be **overloaded** when there are multiple constructors with the *same* name but a *different* signature.
- The parameter list, in the *header* of a constructor or other method, lists the types of the values that are passed and their variable names. These are often referred to as **formal parameters**.
- A parameter is a value that is passed into a constructor or other method. These are often referred to as **actual parameters**.
- Parameters allow values to be passed to the constructor to set the **initial state** of the object.
- The actual parameters passed to a constructor must be compatible with the types identified in the formal parameter list (otherwise, the Java compiler will yell at you!)
- Every object is created using the keyword **new** followed by a call to one of the class's constructors.
- Existing classes and class libraries can be utilized as appropriate to create objects (such as with `String` and `Scanner`!)