public class Instructor {
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
}
Topic 10: Objects

• What is an object? What is a class?
• Using data fields
• Allocating objects
• Instance methods
Why Objects?

As your program gets more complex, you will have lots and lots of variables...

String name;
int score; int age;
String[][] courses;
char[][] table;
Scanner keyboard;
boolean hasBeard;
double posX;
int[][] grid;
Why Objects?

Wouldn't it be nice if you could group those variables together in some way...if you could organize them?

String name;
String[] courses;
int age;
boolean hasBeard;

int score;
int[][] grid;
char[] table;

double posX;

Scanner keyboard;
Why Objects?

We call these groups **Objects**.

String name;
String[] courses;
int age;
boolean hasBeard;

int score;
int[][] grid;
char[] table;

double posX;

Scanner keyboard;
What are Objects?

- An **object** is a variable, where the type is something defined by the programmer.

Primitives – built into Java

Objects – defined by a program

(The Java standard library includes many well-known object types, like **String** and **Scanner**. While they are famous and “standard,” they are still defined by a program.)
What are Objects?

An object can be saved to a variable.

In this example, we are declaring a variable named `russLewis`.

```java
Instructor russLewis =
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
```
What are Objects?

The variables inside an object are its **data fields**. Each can hold a value, like any other variable.

```java
String name = "Russell Lewis";
String[] courses = {"127A","345"};
int age = 37;
boolean hasBeard = true;
```
Why Classes?

Q: Why not just declare all of the fields as local variables?

```java
public static void main(String[] args) {
    String name = "Russell Lewis";
    String[] courses = {"127A","345"};
    int age = 37;
    boolean hasBeard = true;

    ...
}
```
Why Classes?

A: Because you might have several objects with the same fields.

russLewis =
```java
String name = "Russell Lewis";
String[] courses = {"127A","345"};
int age = 37;
boolean hasBeard = true;
```

ericAnson =
```java
String name = "Eric Anson";
String[] courses = {"245","352"};
int age = ???;
boolean hasBeard = false;
```
Why Classes?

We use **classes** to define the fields used by an object. We then define variables using that class as the type.

```java
public class Instructor {
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
}
```

Instructor russLewis = ... ;
Instructor ericAnson = ... ;
What are Classes?

- An **class** is a type defined by a programmer
- Every class is in its own `.java` file
  - File name must match the class name!
- Every **object** has a particular **class**
  - It's impossible to create an object without defining a class first!
public class Instructor
{
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
}

How Do We Define a Class?

For this semester, just assume that every class starts with `public class`.

```java
public class Instructor {
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
}
```
How Do We Define a Class?

public class Instructor {
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
}

This is the name of the class.
It is the name of the type.
public class Instructor
{
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
}
How Do We Define a Class?

When we define a class, every object has its own private copies of every variable.

```java
public class Instructor {
    String name;
    String[] courses;
    int age;
    boolean hasBeard;
}
```

```
russLewis =
String name = "Russell Lewis";
String[] courses = {"127A","345"};
int age = 37;
boolean hasBeard = true;
```

```
ericAnson =
String name = "Eric Anson";
String[] courses = {"245","352"};
int age = ???;
boolean hasBeard = false;
```
Topic 10: Objects

- What is an object? What is a class?
- Using data fields
- Allocating objects
- Instance methods
Using Data Fields

So how do we access the data fields inside an object?

```java
String name = "Russell Lewis";
String[] courses = {"127A","345"];
int age = 37;
boolean hasBeard = true;
russLewis =
```
Using Data Fields

We use the “dot” operator.

```java
russLewis.name
```

```java
String name = "Russell Lewis";
String[] courses = {"127A","345"};
int age = 37;
boolean hasBeard = true;
russLewis =
```
Why Classes?

We can use this syntax to write to fields, or to read from them.

```java
russLewis.name = "mud";
String myName = russLewis.name;
```
Topic 10: Objects

- What is an object? What is a class?
- Using data fields
- Allocating objects
- Instance methods
Allocating Objects

We use \texttt{new} to allocate a new object.

\begin{quote}
\hspace{1cm} Instructor \texttt{russLewis} = \texttt{new Instructor();}
\end{quote}
Allocating Objects

The keyword \texttt{new} is always required.

\begin{verbatim}
Instructor russLewis = new Instructor();
\end{verbatim}
Allocating Objects

The class name is always required.

Instructor russLewis = new Instructor();
Allocating Objects

The parentheses at the end are always required.

Instructor russLewis = new Instructor();

Does this remind you of the syntax for throwing an exception? That's because every exception is an object!
Allocating Objects

When we create a new object, its fields are set to their default values. We must initialize them later, with a different statement.

Instructor russLewis = new Instructor();

russLewis =

String name = null;
String[] courses = null;
int age = 0;
boolean hasBeard = false;
Arrays of Objects

Can we create arrays of objects?
Arrays of Objects

Yes, but with two limitations:
- All must be the same type
- It's a two-step process
Arrays of Objects

Instructor[] department = new Instructor[8];

for (int i=0; i<department.length; i++)
    department[i] = new Instructor();
Arrays of Objects

The array declaration is easy ... same syntax as any other type.

Instructor[] department = new Instructor[8];

for (int i=0; i<department.length; i++)
    department[i] = new Instructor();
Arrays of Objects

Use `new`, just like allocating any other type of array.

Instructor[] department = new Instructor[8];

for (int i=0; i<department.length; i++)
    department[i] = new Instructor();
Arrays of Objects

Instructor[] department = new Instructor[8];

for (int i=0; i<department.length; i++)
    department[i] = new Instructor();

Give the type name.
Arrays of Objects

Give the array size.

Instructor[] department = new Instructor[8];

for (int i=0; i<department.length; i++)
    department[i] = new Instructor();
Arrays of Objects

Note that we do **not** use parentheses!

This is because we are allocating an **array**, not an **object**.

Instructor[] department = new Instructor[8];

for (int i=0; i<department.length; i++)
    department[i] = new Instructor();

If we're technical, an array is a special type of object... but we'll get to that later!
Arrays of Objects

Allocating the array creates memory for the array … but does not create any objects.
As we execute the `for()` loop, we create new objects **one at a time**, and store them in the array slots.
As we execute the `for()` loop, we create new objects **one at a time**, and store them in the array slots.
As we execute the `for()` loop, we create new objects **one at a time**, and store them in the array slots.
Arrays of Objects

As we execute the `for()` loop, we create new objects **one at a time**, and store them in the array slots.

```plaintext
[0] [1] [2] [3] [4] [5] [6] [7]
```
Arrays of Objects

As we execute the `for()` loop, we create new objects **one at a time**, and store them in the array slots.
Arrays of Objects

As we execute the `for()` loop, we create new objects one at a time, and store them in the array slots.
As we execute the `for()` loop, we create new objects one at a time, and store them in the array slots.
Arrays of Objects

As we execute the `for()` loop, we create new objects **one at a time**, and store them in the array slots.
Topic 10: Objects

• What is an object? What is a class?
• Using data fields
• Allocating objects
• Instance methods
Why Instance Methods?

We often find ourselves writing methods which relate to a single object.

```java
String name;
String[] courses;
int age;
boolean hasBeard;

growBeard()
growBeard()

addCourse()
getFirstName()

isAdult()

haveBirthday()
```
Why Instance Methods?

It's often convenient to package commonly-used code inside the object.

String name;
String[] courses;
int age;
boolean hasBeard;

void addCourse(String);
void growBeard();
String getFirstName();
void haveBirthday();
boolean isAdult();
What is an Instance Method?

- An **instance method** is a method that is declared **without** the **static** keyword.

- Instance methods must be called using a particular object.

- The method gets access to the object's fields – as if they were local variables.
Instance Methods

An instance method can access the object's variables like local variables.

```java
public void growBeard()
{
    // this changes the
    // data field to have
    // a new value

    hasBeard = true;
}
```
An instance method looks a lot like methods you've defined before...

```java
public void growBeard()
{
    // this changes the
    // data field to have
    // a new value

    hasBeard = true;
}
```
Instance Methods

One difference is that it must be declared **public** but **not static**.

```java
public void growBeard()
{
    // this changes the
    // data field to have
    // a new value

    hasBeard = true;
}
```
Instance Methods

The other difference is that it can access the fields of the object without using the dot operator.

```java
public void growBeard()
{
    // this changes the
    // data field to have
    // a new value

    hasBeard = true;
}
```
How Do We Define a Class?

public class Instructor
{
    public String name;
    public String[] courses;
    public int age;
    public boolean hasBeard;

    public void growBeard()
    {
        // this changes the
        // data field to have
        // a new value

        hasBeard = true;
    }
}
Instance Methods

So how do we call the instance methods of an object?

```java
public void growBeard()
{
    // this changes the
    // data field to have
    // a new value
    hasBeard = true;
}
```
Instance Methods

The same way you access the data fields!

Instructor russ = ... ;
russ.growBeard();

```java
public void growBeard()
{
    // this changes the
    // data field to have
    // a new value

    hasBeard = true;
}
```
A Couple of Helpful Notes...

```java
public class Instructor {
    public String name;
    public String[] courses;
    public int age;
    public boolean hasBeard;

    public void growBeard() {
        // this changes the
        // data field to have
        // a new value

        hasBeard = true;
    }
}
```

It's easy to forget, but make a habit of marking the methods and the data fields as public.

It won't matter for this semester, but it will matter later on.
public class Instructor
{
    public String name;
    public String[] courses;
    public int age;
    public boolean hasBeard;

    public void growBeard()
    {
        // this changes the
        // data field to have
        // a new value

        hasBeard = true;
    }
}

Notice how every member of this class is non-static?

But every method in our previous classes was static?

It is legal to mix the two in the same class...but we won't do that yet.
Topic 10: Objects

- What is an object? What is a class?
- Using data fields
- Allocating objects
- Instance methods

Summary