“A class … provides a blueprint for the construction of similar objects … you can invent brand new types of your own. Each OO [object-oriented] application gradually becomes a unique programming language… Whether this … ultimately brings you pleasure or gives you pain is a matter of design.”

Sandi Metz, software engineer & author

In Practical Object-Oriented Design in Ruby

Copyright Pearson Education, 2010
Based on slides by Marty Stepp and Stuart Reges
from http://www.buildingjavaprograms.com/
int i = 3;

What is int?
What is i?
What is 3?
```c
int i = 3;
```

What is `int`? The **data type**.

What is `i`? The **variable**.

What is `3`? The **value**.

The type of thing the box is allowed to contain

The name of the box

The thing in the box
- **Declaring** an int variable named i:
  ```
  int i;
  ```

- **Initializing** an int variable named i to 3:
  ```
  int i = 3;
  ```
How about Strings?

- **Declaring** a String variable named str:
  ```java
  String str;
  ```

- **Instantiatiing** a String named str to “ABC”:
  ```java
  String str = "ABC";
  ```

  The type of thing the box is allowed to contain
  The name of the box
  The thing in the box
Why are these different?

```java
int i;
String str;
```
In the computer’s memory

A primitive data type is stored directly in a “house” (memory location):

```
c char c = 'y';  int d = 7;  boolean e = true;
```

 Whereas a String is an “object” that is stored as a reference:

```
String str = "ABC";
```
You can make your own “object” data types!

- See the next few slides....
Nouns & verbs in real life

Classroom
- Array of Students
- Array of Instructors

Student
- String utEid
- doHomework(int assignmentNumber)
- askQuestion(String question)
Object-oriented programming
== nouns & verbs in Java

class Student {
    public String utEid;
    public String major;

    public Student(String utEid) {
        this.utEid = utEid;
    }

    public void askQuestion(String question) {
        System.out.println(question);
    }
}


public class Cs312 {
    public static void main(String[] args) {
        Student[] students = new Student[90];
        student[0] = new Student("joker1234");
        student[1] = new Student("reader5678");
        System.out.println("First student ID is " +
            student[0].utEid);
        System.out.println(getMajor(student[0]));
    }

    public static String getMajor(Student student) {
        if (student.major == null) {
            return "Undeclared";
        }
        return student.major;
    }
}
You can make your own “object” data types!

- Student is a **data type (class)**.
- student[0] and student[1] are **instances** of the Student data type (class).
- student[0] and student[1] are **objects**.
Clicker Question

- What kind of assignment handout do you prefer?
  A. A long assignment handout
  B. A short assignment handout
- Why?
A programming problem

- Given a file of cities' (x, y) coordinates, which begins with the number of cities:
  
  6
  50 20
  90 60
  10 72
  74 98
  5 136
  150 91

- Write a program to draw the cities on a DrawingPanel, then a terrible event (zombie apocalypse, nuclear meltdown) that turns all cities red that are within a given radius:

  Ground zero x: 100
  Ground zero y: 100
  Area of effect: 75
A solution

Scanner input
    = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];
for (int i = 0; i < cityCount; i++) {
    xCoords[i] = input.nextInt();
    yCoords[i] = input.nextInt();
}
...

- **parallel arrays**: 2+ arrays with related data at same indexes.
  - Considered poor style. (Relationship exists in the programmer’s mind, but not explicit in the program.)
Observations

- The data in this problem is a set of points.
- An alternative is to store them as `Point` objects.
  - A `Point` would store a city's x/y data.
  - We could compare distances between `Points` to see whether the terrible event affects a given city.
  - Each `Point` would know how to draw itself.
- The driver program would be shorter and cleaner.
Clients of objects

- **client program**: A program that uses objects.
  - Example: *Zombies* is a client of *DrawingPanel* and *Graphics*.

```java
Zombie.java (client program)
public class Zombie {
    public static void main(String[] args) {
        new DrawingPanel(...)
        new DrawingPanel(...)
        ...
    }
}
```

```java
DrawingPanel.java (class)
public class DrawingPanel {
    ...;
}
```
Classes and objects

- **class**: A program entity that represents either:
  1. A program / module, or
  2. A template for a new type of objects.

  - The `DrawingPanel` class is a template for creating `DrawingPanel` objects.
  - Other classes: `String`, `Random`, `Scanner`, `File`, ...

- **object**: An entity that combines state and behavior.
  - **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects.
Blueprint analogy

iPod blueprint

**state:**
- current song
- volume
- battery life

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song

---

**iPod #1**

**state:**
- song = "1,000,000 Miles"
- volume = 17
- battery life = 2.5 hrs

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song

---

**iPod #2**

**state:**
- song = "Letting You"
- volume = 9
- battery life = 3.41 hrs

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song

---

**iPod #3**

**state:**
- song = "Discipline"
- volume = 24
- battery life = 1.8 hrs

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song
Abstraction

- **abstraction**: A distancing between ideas and details.
  - We can use objects without knowing how they work.

- Abstraction in an iPhone:
  - You understand its external behavior (buttons, screen).
  - You may not understand its inner details, *and you don't need to if you just want to use it.*
Our task

In the following slides, we will implement a `Point` class as a way of learning about defining classes.

- We will define a type of objects named `Point`.
- Each `Point` object will contain x/y data called `fields`.
- Each `Point` object will contain behavior called `methods`.
- Client programs will use the `Point` objects.
**Point objects (desired)**

```java
Point p1 = new Point(5, -2);
Point p2 = new Point(); // origin, (0, 0)
```

- **Data in each `Point` object:**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>the point's x-coordinate</td>
</tr>
<tr>
<td>y</td>
<td>the point's y-coordinate</td>
</tr>
</tbody>
</table>

- **Methods in each `Point` object:**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setLocation(x, y)</td>
<td>sets the point's x and y to the given values</td>
</tr>
<tr>
<td>translate(dx, dy)</td>
<td>adjusts the point's x and y by the given amounts</td>
</tr>
<tr>
<td>distance(p)</td>
<td>how far away the point is from point p</td>
</tr>
<tr>
<td>draw(g)</td>
<td>displays the point on a drawing panel</td>
</tr>
</tbody>
</table>
The class (blueprint) will describe how to create objects.

Each object will contain its own data and methods.
What is output by the following code?

Point p1 = new Point();
Point p2 = new Point();
boolean b1 = (p1 == p2);
System.out.println(b1);

A. Syntax error
B. Runtime error
C. false
D. true
E. no output
Object state:
Fields
public class Point {
    private int x;
    private int y;
}

– Save this code into a file named Point.java.

The above code creates a new type named Point.
– Each Point object contains two pieces of data:
  • an int named x, and
  • an int named y.
– Point objects do not contain any behavior (yet).
**Fields**

- **field**: A variable inside an object that is part of its state.
  - Each object has *its own copy* of each field.

- Declaration syntax:
  ```
  access_modifier type name;
  ```

- Example:
  ```java
  public class Student {
    // each Student object has a name and gpa field (instance variable)
    private String name;
    private double gpa;
  }
  ```
Accessing fields

- Other classes can access/modify an object's fields.
  - depending on the access modifier

- access: `variable.field`
- modify: `variable.field = value;`

Example:

```java
Point p1 = new Point();
Point p2 = new Point();
System.out.println("the x-coord is " + p1.x);     // access
p2.y = 13;                                    // modify
```
A class and its client

- `Point.java` is not, by itself, a runnable program.
  - A class can be used by client programs.

```java
PointMain.java (client program)
public class PointMain {
    public static void main(String[] args) {
        Point p1 = new Point();
        p1.x = 7;
        p1.y = 2;

        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;
        ...
    }
}

Point.java (class of objects)
public class Point {
    int x;
    int y;
}
```

```
x 7 y 2
```

```
x 4 y 3
```
Object behavior: Methods
Client code redundancy

- Suppose our client program wants to draw Point objects:

  ```java
  // draw each city
  Point p1 = new Point();
  p1.x = 15;
  p1.y = 37;
  g.fillOval(p1.x, p1.y, 3, 3);
  g.drawString("(" + p1.x + ", " + p1.y + ")", p1.x, p1.y);
  ```

- To draw other points, the same code must be repeated.
  - We can remove this redundancy using a method.
We can eliminate the redundancy with a static method:

```java
// Draws the given point on the DrawingPanel.
public static void draw(Point p, Graphics g) {
    g.fillOval(p.x, p.y, 3, 3);
    g.drawString("(" + p.x + ", " + p.y + ")", p.x, p.y);
}
```

`main` would call the method as follows:

```java
draw(p1, g);
```
Problems with static solution

- We are missing a major benefit of objects: code reuse.
  - Every program that draws `Points` would need a `draw` method.
- The syntax doesn't match how we're used to using objects.
  ```java
draw(p1, g);  // static (bad)
```
- The point of classes is to combine state and behavior.
  - The `draw` behavior is closely related to a `Point`'s data.
  - The method belongs `inside` each `Point` object.
  ```java
pl.draw(g);    // inside the object (better)
```
Instance methods

- **instance method (or object method):** Exists inside each object of a class and gives behavior to each object.

  ```java
  public type name(parameters) {
    statements;
  }
  ```

  - same syntax as static methods, but without `static` keyword

Example:

```java
public void shout() {
    System.out.println("HELLO THERE!");
}
```
Instance method example

```java
public class Point {
    private int x;
    private int y;

    // Draws this Point object with the given pen.
    public void draw(Graphics g) {
        ...
    }
}
```

- The `draw` method no longer has a `Point p` parameter.
- How will the method know which point to draw?
  - How will the method access that point's x/y data?
Each `Point` object has its own copy of the `draw` method, which operates on that object's state:

```java
Point p1 = new Point(7, 2);
Point p2 = new Point(4, 3);
p1.draw(g);
p2.draw(g);
```

```java
public void draw(Graphics g) {
    // this code can see p1's x and y
}
```

```java
x 7  y 2
```

```java
public void draw(Graphics g) {
    // this code can see p2's x and y
}
```

```java
x 4  y 3
```
The implicit parameter

- **implicit parameter:** The object on which an instance method is called.
  - During the call `p1.draw(g);` the object referred to by `p1` is the implicit parameter.
  - During the call `p2.draw(g);` the object referred to by `p2` is the implicit parameter.
  - The instance method can refer to that object's fields.
    - We say that it executes in the *context* of a particular object.
    - `draw` can refer to the `x` and `y` of the object it was called on.
public class Point {
    int x;
    int y;

    // Changes the location of this Point object.
    public void draw(Graphics g) {
        g.fillOval(x, y, 3, 3);
        g.drawString("(" + x + ", " + y + ")", x, y);
    }
}

- Each Point object contains a draw method that draws that point at its current x/y position.
method questions

- Write a method `translate` that changes a `Point`'s location by a given $dx$, $dy$ amount.

- Write a method `distanceFromOrigin` that returns the distance between a `Point` and the origin, (0, 0).

Use the formula: \[ \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \]

- Modify the `Point` and client code to use these methods.
public class Point {
    int x;
    int y;

    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }

    public double distanceFromOrigin() {
        return Math.sqrt(x * x + y * y);
    }
}