Good question!

We will learn how to create a class that defines a new datatype, i.e., a new type of objects

We need to learn:

1. What is the structure of a class definition?
2. How to specify what happens when an object is instantiated (i.e., when the new operator is used)?
3. How do we define the methods that can be invoked through objects of this class?
1. Structure of class definition

- Data:
  - int acctNumber;
  - double balance;
  - String name;

- Constructor:
  - Account acct1 = new Account("Ted Murphy", 72354, 102.56);
  - Invokes the Account constructor, which is a special method that sets up the object

- Methods:
  - deposit()
  - withdraw()
  - getBalance()
  - toString()

2. Object instantiation

Creating Objects – old example:
- We have already seen something like this:
  - Scanner scan = new Scanner(System.in);
  - This invokes the Scanner constructor, which is a special method that sets up the object

Creating Objects – our newly defined Account class:
- A new Account object is created!
3. Method invocation

- As we have seen, once an object has been created, we can use the dot operator to invoke its methods:

```java
ans = scan.nextLine();
umChars = title.length();
amount = acct1.getBalance();
acct1.deposit (25.85);
```

---

Chapter 4: Writing Classes

- We've been using predefined classes from the Java API. Now we will learn to write our own classes.
  - class definitions
  - constructors
  - instance data
  - method declaration and parameter passing
  - encapsulation and Java modifiers
  - more about creating graphical objects (next week)
```java
public void withdraw (double x, double fee) {
    balance = balance - x - fee;
}
public double getBalance (){
    return balance;
}
public String toString (){
    return acctNumber + "\t" + name + "\t" + fmt.format(balance);
}
```
Creating Account objects

```java
Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
```

Transactions class:

Creating more Account objects

```java
Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
Account acct2 = new Account ("Jane Smith", 69713, 40.00);
```

Account class: Using methods

```java
acct1.deposit (25.85);
```

Transactions class:

Creating Account objects

```java
Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
```

Transactions class:

Creating more Account objects

```java
Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
```

Account class: Using methods

```java
acct1.deposit (25.85);
```
Account class: Another Example

acct1.withdraw (60, 2.50);

public void withdraw (double x, double fee)
{
    balance = balance - x - fee;
}

toString() method

System.out.println(acct1.toString());

public String toString()
{
    NumberFormat fmt = NumberFormat.getCurrencyInstance();
    return acctNumber + ": " + name + " $" + fmt.format(balance);
}

getBalance() method

double amount = acct1.getBalance();

public double getBalance()
{
    return balance;
}
Designing Classes

Class definitions
- A class can contain data declarations and method declarations
  - **Data declarations** (also called **fields**)
  - **Method declarations** (note: the constructor is also a method)

Bank Account Example
- There are some improvements that can be made to the **Account** class
- The design of some methods could also be more robust, such as verifying that the amount parameter to the withdraw() method is positive
- Some of these improvements are in the book examples
  - **Account.java, Transactions.java** (simplified versions)
  - **Account.java, Transactions.java** (book versions)

Examples of datatypes (Classes)

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Name, Address, Grade point average</td>
<td>Set address, Set major, Compute grade point average</td>
</tr>
<tr>
<td>Rectangle</td>
<td>Length, Width, Color</td>
<td>Set length, Set width, Set color</td>
</tr>
<tr>
<td>Aquarium</td>
<td>Material, Length, Weight</td>
<td>Set material, Set length, Set width, Compute volume, Compute density, Compute mass</td>
</tr>
<tr>
<td>Flight</td>
<td>Airline, Right number, Origin city, Destination city, Current status</td>
<td>Set airline, Set flight number, Determine status</td>
</tr>
<tr>
<td>Employee</td>
<td>Name, Department, Title, Salary</td>
<td>Set department, Set title, Compute wages, Compute bonus, Compute taxes</td>
</tr>
</tbody>
</table>

CSC 1051 M.A. Papalaskari, Villanova University

---

```java
//********************************************************************
// RollingDice.java  Author: Lewis/Loftus
// Demonstrates the creation and use of a user-defined class.
//********************************************************************
public class RollingDice {
    public static void main(String[] args) {
        Die die1, die2;
        int sum;
        die1 = new Die();
        die2 = new Die();
        die1.roll();
        die2.roll();
        System.out.println("Die One: "+ die1 +", Die Two: "+ die2);
        die1.roll();
        die2.setFaceValue(4);
        System.out.println("Die One: "+ die1 +", Die Two: "+ die2);
        sum = die1.getFaceValue() + die2.getFaceValue();
        System.out.println("Sum: "+ sum);
        sum = die1.roll() + die2.roll();
        System.out.println("Die One: "+ die1 +", Die Two: "+ die2);
        System.out.println("New sum: "+ sum);
    }
}
```

Sample Run
```
Die One: 5, Die Two: 2
Die One: 1, Die Two: 4
Sum: 5
Die One: 4, Die Two: 2
New sum: 6
```

Another Example: RollingDice.java
Die.java
Author: Lewis/Loftus

Represents one die (singular of dice) with faces showing values between 1 and 6.

public class Die
{
  private final int MAX = 6; // maximum face value
  private int faceValue; // current value showing on the die

  // Constructor: Sets the initial face value.
  public Die()
  {
    faceValue = 1;
  }

  // Rolls the die and returns the result.
  public int roll()
  {
    faceValue = (int)(Math.random() * MAX) + 1;
    return faceValue;
  }

  // Face value mutator.
  public void setFaceValue(int value)
  {
    faceValue = value;
  }

  // Face value accessor.
  public int getFaceValue()
  {
    return faceValue;
  }

  // Returns a string representation of this die.
  public String toString()
  {
    String result = Integer.toString(faceValue);
    return result;
  }
}

---

UML Class Diagrams

UML = Unified Modelling Language

- Example: A UML class diagram for the RollingDice program:

---

UML class diagram for Transactions program?
NEXT: Focus on Methods

- Method control flow
- Method definition
  - Parameters
  - return statement
- Different ways of thinking about methods

Method Control Flow

- If the called method is in the same class, only the method name is needed

Method Control Flow

- The called method is often part of another class or object
- Thus the dot operator is an addressing mechanism. Note that it can also be used to access an object’s or class’s data directly, for example
  - acct1.name
  - Color.black
- more on this later (encapsulation)

Invoking methods within the same class

- An object’s method may access any of the object’s other methods directly. Eg:

  ```java
  public void addInterest(double rate) {
    deposit (rate*balance);
  }
  ```

  Client code, eg: compound the interest for acct2 over 10 years
  ```java
  int year = 1;
  while (year <= 10) {
    acct2.addInterest(0.03*balance);
    year ++;
  }
  ```
Designing Classes

Method Control Flow: example

Method definition: Example

- parameters
- return type
- return statement

```java
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);
    return result;
}
```

More Method Examples:

- Write a method with two double parameters \( a \) and \( b \) that computes and returns the sum of squares of its two parameters (i.e., \( a^2 + b^2 \)).

More Method Examples:

- Write a method with one int parameter \( num \), that returns a String composed of “Happy Birthday” \( num \) times.

How do we invoke the method to compute & print: \((14.8)^2 + (37.65)^2\) ?

How do we invoke the method to print “happy birthday” 4 times?
Getting to know classes so far

- Using predefined classes from the Java API.
- Defining classes for our own datatypes.

**datatypes:**
- Account
- Die
- Shoe
- Person

**Clients (Driver classes):**
- Transactions, OnePercent
- RollingDice
- YouVeGotShoes (Project)
- PeopleBeingPeople (Lab)

Next: visibility modifiers, graphical objects,

---

Encapsulation

- An encapsulated object can be thought of as a *black box* -- its inner workings are hidden from the client
- The client invokes the interface methods which in turn manage the instance data

**Violating Encapsulation**
It is possible for a class to access the instance data of another class directly

**Use Accessors & Mutators - RIGHT**
- Indirect access through methods
- accessors and mutators (“getters” and “setters”)
- Usually named getX() and setX()

---

**Violating Encapsulation - WRONG**
- It is possible for a class to access the instance data of another class directly – *but it’s not a good idea!*
Visibility Modifiers

- In Java, we enforce encapsulation through the appropriate use of visibility modifiers:
  - `public` – can be referenced from other classes
  - `private` – can be referenced only within that class:
  - `protected` – involves inheritance (discussed later)
- Data declared without a visibility modifier have default visibility and can be referenced by any class in the same package
- An overview of all Java modifiers is presented in Appendix E

Violating Encapsulation experiment

- Revisit the Account example
  (use our simplified versions from the course website or Lab 8)
- Add some code to the client (Transactions or Onepercent) to modify the value of an instance variable, eg:
  ```java
  acct1.name = "Bernie";
  ```
- Run the program to verify that the name on that account has changed.
- Now modify Account.java – insert the modifier `private` in front of the instance variable declaration:
  ```java
  private String name;
  ```
- Without changing anything in the client, re-compile the Account class and run your program again. Note the error you get:
  Error: ____________________________________________________

Public Constants... OK

Example: The Account class can have a constant for the interest rate:

```java
public final double RATE = 0.015;
```

A client (eg, OnePercent.java) can access this constant directly:

```java
System.out.print("Interest rate = " + acct1.RATE);
```

`static` Public Constants... BETTER

It is better to declare constants as `static`

```java
public final static double RATE = 0.015;
```

This way, a client can access the constants without creating an object, using the class name:

```java
System.out.print("Interest rate = " + Account.RATE);
```
Visibility Modifiers – the **RULES**

<table>
<thead>
<tr>
<th></th>
<th>public</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>NO (but OK for public constants)</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods</td>
<td>Yes</td>
<td>Yes, for support methods only</td>
</tr>
</tbody>
</table>

See also: [ImInUrClassMessingUrInstanceData.java](#)

---

### Graphical Objects

- Some objects contain information that determines how the object should be represented visually.
- **Graphical objects**
  - data about position, size, and other attributes
  - methods to draw the object

- Let’s look at some other examples of graphical objects:
  - **Example 1:** SmilingFacePanel
    - SmilingFace.java
    - SmilingFacePanel.java
    - Splat.java
    - SplatPanel.java
    - Circle.java
  - **Example 2:** SplatPanel

```java
//********************************************************************
//  SmilingFace.java       Author: Lewis/Loftus
//  Demonstrates the use of a separate panel class.
//********************************************************************
import javax.swing.JFrame;
public class SmilingFace
{
  //-----------------------------------------------------------------
  //  Creates the main frame of the program.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    JFrame frame = new JFrame ("Smiling Face");
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    SmilingFacePanel panel = new SmilingFacePanel();
    frame.getContentPane().add(panel);
    frame.pack();
    frame.setVisible(true);
  }
}
```
The SmilingFace program draws a face by defining the paintComponent method of a panel.

A UML class diagram:

```
SmilingFace --> SmilingFacePanel
  BASEX: int
  BASEY: int
  paintComponent(p: Graphics): void
```

```java
import javax.swing.JPanel;
import java.awt.*;
public class SmilingFacePanel extends JPanel {
    private final int BASEX = 120, BASEY = 60; // base point for head
    public SmilingFacePanel () {
        setBackground (Color.blue);
        setPreferredSize (new Dimension(320, 200));
        setFont (new Font("Arial", Font.BOLD, 16));
    }
    public void paintComponent (Graphics page) {
        super.paintComponent (page);
        page.setColor (Color.yellow);
        page.fillOval (BASEX, BASEY, 80, 80); // head
        page.fillOval (BASEX-5, BASEY+20, 90, 40); // ears
        page.setColor (Color.black);
        page.drawOval (BASEX+20, BASEY+30, 15, 7); // eyes
        page.drawOval (BASEX+45, BASEY+30, 15, 7);
        page.fillOval (BASEX+25, BASEY+31, 5, 5); // pupils
        page.drawArc (BASEX+20, BASEY+25, 15, 7, 0, 180); // eyebrows
        page.drawArc (BASEX+45, BASEY+25, 15, 7, 0, 180);
        page.drawArc (BASEX+35, BASEY+40, 15, 10, 180, 180); // nose
        page.drawString ("Always remember that you are unique!",
                        BASEX-105, BASEY-15);
        page.drawString ("Just like everyone else.", BASEX-45, BASEY+105);
    }
```

Jpanel Class – let’s look at the Java API

The SmilingFacePanel class is derived from the JPanel class using inheritance.

We are defining a subclass of JPanel.

CSC 1051 M.A. Papalaskari, Villanova University
Smiling Face Example

- Every Swing component has a `paintComponent` method.
- The `paintComponent` method accepts a `Graphics` object that represents the graphics context for the panel.
- We define the `paintComponent` method to draw the face with appropriate calls to the `Graphics` methods.
- Note the difference between drawing on a panel and adding other GUI components to a panel.

Objects with a draw() method

- The next example - Splat - is structured differently.
- It draws a set of colored circles on a panel, but each circle is represented as a separate object that maintains its own graphical information.
- The `paintComponent` method of the panel "asks" each circle to draw itself.
- See [Splat.java](Splat.java)
- See [SplatPanel.java](SplatPanel.java)
- See [Circle.java](Circle.java)

```java
// Demonstrate the use of graphical objects.
import javax.swing.*;
import java.awt.*;
public class Splat {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Splat");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new SplatPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
```
Splat

main (args : String[]): void

SplatPanel

circle1: Circle
circle2: Circle
circle3: Circle
circle4: Circle
circle5: Circle

paintComponent(p: Graphics): void

Circle
diameter: int
color: Color
x: int
y: int
draw(p: Graphics): void
getDiameter(): int
color(): Color
getX(): int
getY(): int

getDiameter(): int
color(): Color
getX(): int
getY(): int
setDiameter(size: int): void
color(shade: Color): void
setX(upperX: int): void
setY(upperY: int): void

//********************************************************************
//  SplatPanel.java       Author: Lewis/Loftus
//  Demonstrates the use of graphical objects.
//********************************************************************

import javax.swing.*;
import java.awt.*;

public class SplatPanel extends JPanel {
    private Circle circle1, circle2, circle3, circle4, circle5;

    public SplatPanel() {
        circle1 = new Circle(30, Color.red, 70, 35);
circle2 = new Circle(50, Color.green, 30, 20);
circle3 = new Circle(100, Color.cyan, 60, 85);
circle4 = new Circle(45, Color.yellow, 170, 30);
circle5 = new Circle(60, Color.blue, 200, 60);
        setPreferredSize(new Dimension(300, 200));
        setBackground(Color.black);
    }

    public void paintComponent(Graphics page) {
        super.paintComponent(page);
circle1.draw(page);
circle2.draw(page);
circle3.draw(page);
circle4.draw(page);
circle5.draw(page);
    }
}

continue

CSC 1051 M.A. Papalaskari, Villanova University
import java.awt.*;
public class Circle {
    private int diameter, x, y;
    private Color color;

    public Circle (int size, Color shade, int upperX, int upperY) {
        diameter = size;
        color = shade;
        x = upperX;
        y = upperY;
    }

    public void draw (Graphics page) {
        page.setColor (color);
        page.fillOval (x, y, diameter, diameter);
    }

    public void setDiameter (int size) {
        diameter = size;
    }

    public void setColor (Color shade) {
        color = shade;
    }

    public void setX (int upperX) {
        x = upperX;
    }

    public void setY (int upperY) {
        y = upperY;
    }

    public int getDiameter () {
        return diameter;
    }

    public Color getColor () {
        return color;
    }

    public int getX () {
        return x;
    }

    public int getY () {
        return y;
    }
}