Objects and Classes

CSE160: Computer Science A: Honors
Paul Fodor
Stony Brook University
http://www.cs.stonybrook.edu/~cse160
Motivating Problems

- Develop a Graphical User Interface (GUI)
  - need of multiple object instances of classes

- Relational databases:
  - several tuples of the same relation schema
    - Example: Person(firstName, lastName, Address, dateOfBirth)

- 2 buttons
- input fields
- 2 check boxes
- 2 radio/choice boxes
- lists
An object represents an entity in the real world that can be distinctly identified from a class of objects with common properties.

An object has a unique state and behavior:
- the state of an object consists of a set of data fields (properties) with their current values
- the behavior of an object is defined by a set of instance methods
Classes

• In Java **classes** are templates that define objects of the same type

  • A Java class uses:
    • non-static/instance variables to define data fields
    • non-static/instance methods to define behaviors

  • A class provides a special type of methods called **constructors** which are invoked to construct objects from the class
class Circle {
    /** The radius of this circle */
    private double radius = 1.0;

    /** Construct a circle object */
    public Circle() {
    }

    /** Construct a circle object */
    public Circle(double newRadius) {
        radius = newRadius;
    }

    /** Return the area of this circle */
    public double getArea() {
        return radius * radius * 3.14159;
    }
}

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
public class TestCircle {

    public static void main(String[] args) {

        Circle c1 = new Circle();
        Circle c2 = new Circle(5.0);

        System.out.println( c1.getArea() );
        System.out.println( c2.getArea() );

    }

}
The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a object-oriented system.

UML Class Diagram

- **Class name**: Circle
- **Data fields**: radius: double
- **Constructors and methods**:
  - Circle()
  - Circle(newRadius: double)
  - getArea(): double

- **circle1**: Circle
  - radius = 1.0

- **circle2**: Circle
  - radius = 25

- **circle3**: Circle
  - radius = 125
Constructors

• Constructors must have the same name as the class itself.
• Constructors do not have a return type—not even `void`.
• Constructors are invoked using the `new` operator when an object is created – they initialize objects to reference variables:

```java
ClassName o = new ClassName();
```

• Example:

```java
Circle myCircle = new Circle(5.0);
```

• A class may be declared without constructors: a no-arg `default constructor` with an empty body is `implicitly` declared in the class
Accessing Objects

- Referencing the object’s data:
  \[
  \text{objectRefVar.data}
  \]
  
  Example: \(\text{myCircle.radius}\)

- Invoking the object’s method:
  \[
  \text{objectRefVar.methodName(arguments)}
  \]
  
  Example: \(\text{myCircle.getArea()}\)
Using classes

Circle myCircle = new Circle(5.0);

SCircle yourCircle = new Circle();

yourCircle.radius = 100;
Using classes

Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
Using classes

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

Assign object reference to myCircle

myCircle

reference value

: Circle

radius: 5.0
Using classes

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
Using classes

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

Create a new Circle object

myCircle

reference value

: Circle
radius: 5.0

yourCircle
null value

: Circle
radius: 1.0
Using classes

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

Assign object reference to yourCircle

myCircle

: Circle
radius: 5.0

reference value

yourCircle

: Circle
radius: 1.0

reference value
Using classes

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

Change radius in yourCircle
Static vs. Non-static variables

- Static variables and constants:
  - global variables for the entire class: for all objects instances of this class

```java
static int count = 0;
static final double PI = 3.141592;
```

- Non-static/instance variables are date fields of objects:

```java
System.out.println(myCircle.radius);
System.out.println(yourCircle.radius);
```
Static Variables, Constants and Methods

- Static variables are shared by all the instances of the class:

```
Circle
  -radius: double
  -numberOfObjects: int
  +getNumberOfObjects(): int
  +getArea(): double
```

After two Circle objects were created, `numberOfObjects` is 2.

UML Notation:
- `+`: public variables or methods
- `_`: static variables or methods

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Static vs. Non-static methods

- Static methods:
  - Shared by all the instances of the class - not tied to a specific object:
    ```java
double d = Math.pow(3, 2);
```
- Non-static/instance methods must be invoked from an object instance of the class:
  ```java
double d1 = myCircle.getArea();
double d2 = yourCircle.getArea();
```
No Default values for local variables

Java assigns no default value to a local variable inside a method.

```java
public class Test {
    public static void main(String[] args) {
        int x; // x has no default value
        String y; // y has no default value
        System.out.println("x is " + x);
        System.out.println("y is " + y);
    }
}
```

**Compilation errors:** the variables are not initialized
Default values for Data Fields

- Data fields have default values

- Example:

```java
public class Student {
    String name; // name has default value null
    int age; // age has default value 0
    boolean isScienceMajor; // isScienceMajor has default value false
    char gender; // c has default value 'u0000'
}

public class Test {
    public static void main(String[] args) {
        Student student = new Student();
        System.out.println("name? " + student.name); // null
        System.out.println("age? " + student.age); // 0
        System.out.println("isScienceMajor? " + student.isScienceMajor); // false
        System.out.println("gender? " + student.gender); //
    }
}
```

Note: If a data field of a reference type does not reference any object, the data field holds a special literal value: `null`. 
Differences between Variables of Primitive Data Types and Object Types

**Primitive type**

```
int i = 1        i
```

1

**Object type**

```
Circle c       c reference
```

Created using `new Circle()`

```
c: Circle
radius = 1
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Copying Variables of Primitive Data Types and Object Types

**Primitive type assignment** \( i = j \)

Before:

\[
\begin{array}{c}
1 \\
2 \\
2 \\
\end{array}
\]

After:

\[
\begin{array}{c}
2 \\
2 \\
2 \\
\end{array}
\]

**Object type assignment** \( c1 = c2 \)

Before:

\[
\begin{array}{c}
\text{c1: Circle} \\
radius = 5 \\
\text{c2: Circle} \\
radius = 9 \\
\end{array}
\]

After:

\[
\begin{array}{c}
\text{c1: Circle} \\
radius = 5 \\
\text{c2: Circle} \\
radius = 9 \\
\end{array}
\]
Garbage Collection

• The object previously referenced by c1 is no longer referenced, it is called garbage

• Garbage is automatically collected by the JVM, a process called garbage collection

• In older languages, like C and C++, one had to explicitly deallocate/delete unused data/objects
### Example classes in Java: the Date class

Java provides a system-independent encapsulation of date and time in the **java.util.Date** class.

The **toString** method returns the date and time as a string.

<table>
<thead>
<tr>
<th>java.util.Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Date()</td>
<td>Constructs a Date object for the current time.</td>
</tr>
<tr>
<td>+Date(elapseTime: long)</td>
<td>Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.</td>
</tr>
<tr>
<td>+toString(): String</td>
<td>Returns a string representing the date and time.</td>
</tr>
<tr>
<td>+getTime(): long</td>
<td>Returns the number of milliseconds since January 1, 1970, GMT.</td>
</tr>
</tbody>
</table>

January 1, 1970, GMT is called the Unix time (or Unix epoch time).

```java
java.util.Date date = new java.util.Date();
System.out.println(date.toString());
```
## The Random class

**java.util.Random**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Random()</td>
<td>Constructs a Random object with the current time as its seed.</td>
</tr>
<tr>
<td>+Random(seed: long)</td>
<td>Constructs a Random object with a specified seed.</td>
</tr>
<tr>
<td>+nextInt(): int</td>
<td>Returns a random int value.</td>
</tr>
<tr>
<td>+nextInt(n: int): int</td>
<td>Returns a random int value between 0 and n (exclusive).</td>
</tr>
<tr>
<td>+nextLong(): long</td>
<td>Returns a random long value.</td>
</tr>
<tr>
<td>+nextDouble(): double</td>
<td>Returns a random double value between 0.0 and 1.0 (exclusive).</td>
</tr>
<tr>
<td>+nextFloat(): float</td>
<td>Returns a random float value between 0.0F and 1.0F (exclusive).</td>
</tr>
<tr>
<td>+nextBoolean(): boolean</td>
<td>Returns a random boolean value.</td>
</tr>
</tbody>
</table>

```java
Random random1 = new Random(3);
for (int i = 0; i < 10; i++)
    System.out.print(random1.nextInt(1000) + " ");
```

734 660 210 581 128 202 549 564 459 961
Visibility Modifiers and Accessor/Mutator Methods

• By default, the class, variable, or method can be accessed by any class in the same package.
  - public (+ in UML)
    The class, data, or method is visible to any class in any package.
  - private (- in UML)
    The data or methods can be accessed only by the declaring class - To protect data!

• getField (accessors) and setField (mutators) methods are used to read and modify private properties.
The `private` modifier restricts access to within a class
The default modifier restricts access to within a package
`public` – unrestricted access

```java
package p1;

class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}

package p2;

public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;
    
    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}

package p1;

class C1 {
  ...
}

package p2;

public class C2 {
  can access C1
}

package p1;

public class C3 {
  cannot access C1;
  can access C2;
}
```

Packages and modifiers
## UML: Data Field Encapsulation

### Circle

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-radius: double</td>
<td>The radius of this circle (default: 1.0).</td>
</tr>
<tr>
<td>-numberOfObjects: int</td>
<td>The number of circle objects created.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Circle()</td>
<td>Constructs a default circle object.</td>
</tr>
<tr>
<td>+Circle(radius: double)</td>
<td>Constructs a circle object with the specified radius.</td>
</tr>
<tr>
<td>+getRadius(): double</td>
<td>Returns the radius of this circle.</td>
</tr>
<tr>
<td>+setRadius(radius: double): void</td>
<td>Sets a new radius for this circle.</td>
</tr>
<tr>
<td>+getNumberOfObject(): int</td>
<td>Returns the number of circle objects created.</td>
</tr>
<tr>
<td>+getArea(): double</td>
<td>Returns the area of this circle.</td>
</tr>
</tbody>
</table>

The - sign indicates private modifier.
Array of Objects

- An array of objects is an array of reference variables (like the multi-dimensional arrays seen before)

```java
Circle[] circleArray = new Circle[10];
circleArray[0] = new Circle();
circleArray[1] = new Circle(5);
...
```

![Diagram showing an array of objects with references to Circle objects]