Objects and Classes

CSE 114, Computer Science 1
Stony Brook University
http://www.cs.stonybrook.edu/~cse114
Opening Problem

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

- 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 methods

### Class Name: Circle

| Data Fields: radius is ________ |
| Methods: getArea() |

### Circle Object 1

Data Fields:
- radius is 10

### Circle Object 2

Data Fields:
- radius is 25

### Circle Object 3

Data Fields:
- radius is 125

A class template

Three objects of the Circle class
Classes

- Classes are templates that define objects of the same type
- A Java class uses:
  - non-static variables to define data fields and
  - non-static methods to define behaviors
- A class provides a special type of methods called **constructors** which are invoked to construct objects from the class
```java
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;
    }
}
```
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, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.
• UML Class Diagram:
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:

  ```
  ClassName o = new ClassName();
  ```
- Example:

  ```
  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:
  \texttt{objectRefVar.data}
  
  - Example: \texttt{myCircle.radius}

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

```java
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;

Create a circle
Using classes

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

Assign object reference to myCircle

myCircle

reference value

radius: 5.0

: Circle
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
Using classes

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

Assign object reference to yourCircle

myCircle

reference value

: Circle

radius: 5.0

yourCircle

reference value

: Circle

radius: 1.0
Using classes

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

Change radius in yourCircle
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, 2.5);
    ```

  • Non-static/instance methods must be invoked from an object instance of the class:
    
    ```java
    double d1 = myCircle.getArea();
    double d2 = yourCircle.getArea();
    ```
Default values

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

BUT it assigns default values to data fields!
Reference Data Fields

• Data fields have default values

• Example:

```java
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'
}
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); //
    }
}
```

• 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
```

Object type
```
Circle c
```

Created using new Circle()
```
c: Circle
radius = 1
```
Copying Variables of Primitive Data Types and Object Types

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

Before:

\[
\begin{array}{c}
  i \\
  j
\end{array}
\]

After:

\[
\begin{array}{c}
  i \\
  j
\end{array}
\]

**Object type assignment** \( c_1 = c_2 \)

Before:

\[
\begin{array}{c}
  c_1 \\
  c_2
\end{array}
\]

\[
\begin{array}{c}
  c_1: \text{Circle} \\
  \text{radius} = 5
\end{array}
\]

\[
\begin{array}{c}
  c_2: \text{Circle} \\
  \text{radius} = 9
\end{array}
\]

After:

\[
\begin{array}{c}
  c_1 \\
  c_2
\end{array}
\]

\[
\begin{array}{c}
  c_1: \text{Circle} \\
  \text{radius} = 5
\end{array}
\]

\[
\begin{array}{c}
  c_2: \text{Circle} \\
  \text{radius} = 9
\end{array}
\]
Garbage Collection

- The object previously referenced by c1 is no longer referenced, it is called garbage
- Garbage is automatically collected by JVM
  - 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 \texttt{java.util.Date} class.

The \texttt{toString} method returns the date and time as a string.

The + sign indicates public modifier

<table>
<thead>
<tr>
<th>Method</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).

\begin{verbatim}
java.util.Date date = new java.util.Date();
System.out.println(date.toString());
\end{verbatim}
The Random class

java.util.Random

+Random()
+Random(seed: long)
+nextInt(): int
+nextInt(n: int): int
+nextLong(): long
+nextDouble(): double
+nextFloat(): float
+nextBoolean(): boolean

Constructs a Random object with the current time as its seed.
Constructs a Random object with a specified seed.
Returns a random int value.
Returns a random int value between 0 and n (exclusive).
Returns a random long value.
Returns a random double value between 0.0 and 1.0 (exclusive).
Returns a random float value between 0.0F and 1.0F (exclusive).
Returns a random boolean value.

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
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
underline: static variables or methods
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!

• get and set methods are used to read and modify private properties.
Packages

- The **private** modifier restricts access to **within a class**
- The default modifier restricts access to **within a package**
- **public** – unrestricted access

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

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

package p1;
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 p2;
public class C3 {
  void aMethod() {
    p1.C1 o = new p1.C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;
    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}

class C1 {
  ...
}

package p1;
public class C2 {
  can access C1
}

package p2;
public class C3 {
  cannot access C1;
  can access C2;
}
```
The - sign indicates private modifier.

<table>
<thead>
<tr>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-radius: double</td>
</tr>
<tr>
<td>-numberOfObjects: int</td>
</tr>
<tr>
<td>+Circle()</td>
</tr>
<tr>
<td>+Circle(radius: double)</td>
</tr>
<tr>
<td>+getRadius(): double</td>
</tr>
<tr>
<td>+setRadius(radius: double): void</td>
</tr>
<tr>
<td>+getNumberOfObject(): int</td>
</tr>
<tr>
<td>+getArea(): double</td>
</tr>
</tbody>
</table>

The radius of this circle (default: 1.0).
The number of circle objects created.

Constructs a default circle object.
Constructs a circle object with the specified radius.
Returns the radius of this circle.
Sets a new radius for this circle.
Returns the number of circle objects created.
Returns the area of this circle.
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();
...
```

Circle object 0

Circle object 1

Circle object 9