

Define and use a simple class

- First version of GradeBook.java (Fig. 3.1, p. 75)

```
public class GradeBook {
    public void displayMessage() {
        System.out.println ( "Welcome ..." );
    }
}
```

- First GradeBookTest.java (Fig. 3.2, p. 77)

```
public class GradeBookTest {
    public static void main( String args[] ) {
        GradeBook myGradeBook = new GradeBook();
        myGradeBook.displayMessage();
    }
}
```

- Notice all GradeBook objects are exactly the same

Instance variables

- Each object is an **instance** of its class, and each instance can have different attributes
- e.g., course name for GradeBook object:

```
private String courseName;
```
- Related *set* and *get* methods:

```
public void setCourseName(String name)
{ courseName = name; }

public String getCourseName()
{ return courseName; }
```
- See enhanced [GradeBook.java](#) (Fig. 3.7, p. 83) and new [GradeBookTest.java](#) (Fig. 3.8)
 - Notice name is `null` before set method is used
 - Numeric values default to 0 & boolean values to `false`

Constructors

- Definition looks like a method, but ... always has same name as the class, and no return type

- e.g., alternate constructor for GradeBook:

```
public GradeBook(String name)
{   courseName = name; }
```

- Initialize course name as object constructed:

```
GradeBook myBook =
    new GradeBook ( "CS 5JA" );
```

- No need to set later, and never equals null

- See another [GradeBook.java](#) (Fig. 3.10) and another new [GradeBookTest.java](#) (Fig. 3.11)

Syntax for defining methods

- Method has two parts – a header and a body

```
type name (parameter declarations) // header  
{ local declarations and statements } // body
```

- Parentheses in header and brackets around body are required
- **type** – refers to the result of the method
 - May be any primitive type, or any class
 - Or may be `void` – means it does not return any results
- If not `void`, statements in the method body *must* include a `return` statement

Java has 8 primitive data types (everything else is an object)

- 7 are “number” types
 - 5 of the number types are *integral* types:
 - `int` – most fundamental; 4, -123, 9587123 are `int`
 - `long` – for longer integers (>2,147,483,647)
 - `short`, `byte` – save space for shorter integers
 - `char` – to represent characters; ‘A’, ‘a’, ‘\n’
 - Other 2 number types are *floating point* types:
 - `double` – most fundamental; 0.4, -123.3, 95.
 - `float` – save space for less precision
- 8th type is `boolean`: to represent `true` or `false`

About floating point types

- *Rounding errors* occur when an exact conversion between numbers is not possible

```
double f = 4.35;
```

```
System.out.println(100 * f); // prints 434.99999999999994
```

- Illegal to assign a floating-point expression to an integer

```
double balance = 13.75;
```

```
int dollars = balance; // Error
```

- **Casts**: used to convert a value to a different type

```
int dollars = (int) balance; // OK
```

- Cast discards fractional part – *truncates*

- `Math.round` converts floating-point to nearest integer

- `long rounded = Math.round(balance);`

- If balance is 13.75, then rounded is set to 14