Chapter 3

Classes

Lesson page 3-1. Classes

Activity 3-1-1 The class as a file drawer of methods

Question 1. The form of a class definition is:

> **public class** <*class name*> {
>   <*definitions of static methods*>
> }

Question 2. A class does not get executed. A class is simply a container, like a drawer of a filing cabinet.

Question 3. Static methods defined in the class are kept in the file-cabinet drawer for that class.

Question 4. Keyword **public** is an access modifier. Its presence as a prefix of a class, method, or field definition indicates that the entity is visible to (can be accessed by) all parts of the program.

Question 5. A Java program consists of a collection of classes.

Question 6. Each class definition is placed in its own file. The class names begin with capital letters. The file that contains a class `X` is named `X.java`. For example, the file that contains class `Foo` is named `Foo.java`.

Activity 3-1-2 Referencing static methods

Question 7. Here are two rules for referring to a static method:

1. To refer to static method `M` defined in class `C`, use the basic form `C.M`.
2. Within class `C` itself, one can eliminate the "`C.`" and use simply `M`.

Activity 3-1-3 Temperature conversion

Question 8. Goal (1), having a correct program, is most important. However, correctness is best achieved by having (3) a simple and readable progam, so (3) is of second importance. Whether one chooses to (2) minimize execution time or (4) minimize programming time next depends entirely on the context. In some professional programming, execution speed is of paramount importance, and might even take priority over (3), having a simple program. In the context of anyone using ProgramLive, minimizing programming time should generally take precedence over minimizing execution speed.

Question 9. We relied on the first three methods to do all the conversions.

Activity 3-1-4 Overloading method names

Question 10. Overloading occurs when two methods have the same name. If two methods have the name `zot`, `zot` is overloaded.

Question 11. To determine which method is called when there is overloading, Java uses

1. the kind of method (function or procedure), and
2. the number and types of the arguments.

Lesson page 3-2. Class `Math`

Activity 3-2-1 The static methods in class `Math`

Question 1. Here are the specifications:

- `abs(x)` yields the absolute value of its parameter.
- `floor(x)` yields the largest integer that is not greater than its argument.
- `ceil(x)` yields the smallest integer that is not less than its argument.
- `min(x,y)` yields the smallest of `x` and `y`.
- `max(x,y)` yields the largest of `x` and `y`.

Question 2. The expression has the value `1.0`.

Question 3. The expression is `Math.abs(-100 + Math.min(a,b))`.

Activity 3-2-2 The constants in class `Math`

Question 4. A static variable is a variable declared in a class with modifier **static**. "static field" and "static variable" are synonyms.

Question 5. A static field is a variable declared in a class with modifier **static**. "static field" and "static variable" are synonyms.

Question 6. A class variable is a static field. "Class variable" and "static field" are synonyms.

Question 7. In Java, a constant is a variable that is declared with modifier final.

Question 8. The value `pi` is the ratio of the diameter of a circle to its circumference.

Question 9. Remembering that 1 raised to any power equals 1, we write the declaration: **double** `area= Math.PI;`

Question 10. This is declared as **final** so it can't be changed:
public static final double `EARTH_WEIGHT= 6.0E24;`

Activity 3-2-3 Exercises on static methods and variables

Question 11. Here is class `Small`:

```
public class Small {
    public static void main(String[] args) {
        System.out.println(Stats.eW);
    }
}
```

Activity 3-2-4 Trigonometric functions

Lesson page 3-3. Classes and objects

Activity 3-3-1 Using an object to aggregate information

Question 1. An object is like a manila folder that contains various pieces of information. In ProgramLive, we draw objects as manila folders. The name of the object is on the tab of the folder.

Question 2. To aggregate means to bring together, to collect in one sum, mass, or body.

Question 3. *Field* is a synonym for *instance variable.* An instance variable is a field of an object, i.e. a non-static variable that is defined in a class and thus is a component of every instance (or object) of the class.

Activity 3-3-2 Referencing an object and its fields

Question 4. Refer to field `x` of object `obj` using: `obj.x` .

Question 5. This assignment makes `s` contain the same name as `r`: `s= r;`

Activity 3-3-3 Definition of a class

Question 6. The format of a class definition is:

```
public class <class name> {
  <definitions of methods and variables>
}
```

Question 7. A declaration of a field looks like:

```
public <type> <identifier>;
```

or, more generally like:

<modifier> <type> <identifier>;

Question 8. The field that belongs to all objects of that class (with each object having its own field) is declared without keyword **static**. A variable that belongs to the class itself is declared **static**.

Question 9. Objects are manila folders, and they are placed in the file-cabinet drawer for their class.

Question 10. False. A nonstatic variable belongs in the manila folder of each object of the class.

Question 11. Here is the definition of class `Animal`

```
public class Animal {
    public double weight;
    public int numberOfLegs;
    public String color;
}
```

Activity 3-3-4 A class as a type

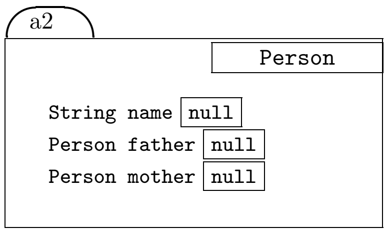
Question 12. A type is a set of values together with operations on them.

Question 13. A primitive type is a type that is built in to Java. The primitive types are: **boolean**, **byte**, **char**, **double**, **float**, **int**, **long**, and **short**.

Question 14. Any Java class is a class type.

Question 15. The constant **null** denotes the absence of an object name, so if **null** is in variable `v`, `v` does not contain the name of an object.

Question 16. The declaration of variable `Sam` is `Person Sam;` . Here is an object of class `Person`:



Lesson page 3-4. Creating and initializing objects

Activity 3-4-1 Creating objects

Question 1. The form a `new` expression is:

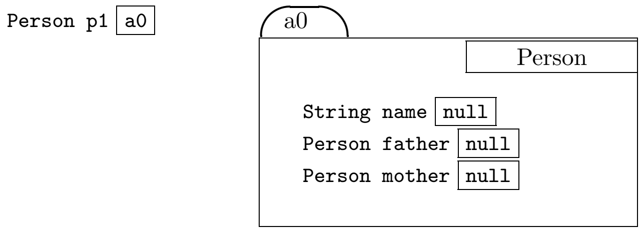
new <*class name*> ( )

Later, we will generalize this form.

Question 2. To evaluate a `new` expression, create a new object of class <*class name*>, place the object in the file drawer for that class, and yield the name of the new object.

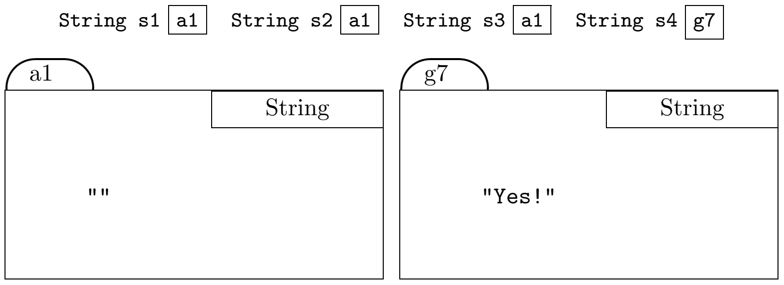
Question 3. *Instance* of a class and *object* of a class are synomnyms.

Question 4. Below, we show the contents of `p1` and the object after execution of this assignment.



Activity 3-4-2 Creating `String` objects

Question 5. Below, we show the contents of the variables after execution of the assignments.



Question 6. `String s= "Elaine";`
`String s= new String("Elaine");`

Question 7. Eventually, Java performs "garbage collection"; it looks through memory for locations that the program is not using and makes them usable again. An object to which no variable refers is a candidate for such garbage collection.

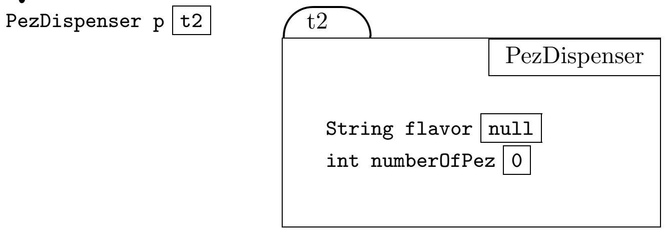
Activity 3-4-3 Using field declarations to initialize fields

Question 8.

- **byte**, **short**, **int**, **long**: 0
- **float**, **double**: 0.0

- **boolean**: **false**
- **char**: the null character, '\u0000'
- Class-type: `null`

Question 9.



Question 10. Here is the revised class:

```
// Container for Pez, a type of candy.
public class PezDispenser {
    String flavor= "";   // Flavor of this Pez container
    int numberOfPez= 20; // No. candies in this container
}
```

Question 11. The inital value of `sq1` is `null`. This means that `sq1` doesn't contain a referece to an object. Variable `sq1` may contain the name of any instance of class `Animal`.

Question 12.

```
// An Animal holds basic information about an animal
public class Animal {
    String name;       //the name of this Animal
    String species;    //its species
    int age;           //its age, in years
    boolean hasShots; // = "the animal has had shots"
}
```

Lesson page 3-5. Scope boxes and constructors

Activity 3-5-1 The scope box of a frame

Question 1. The scope contains either the name of an instance of a class or the name of a class.

Question 2. The scope box of a frame indicates where to look for a name if it is not found in the frame.

Activity 3-5-2 The constructor

Question 3. A constructor has neither **void** nor a type in its header, and its name is the same as the name of the class in which it appears.

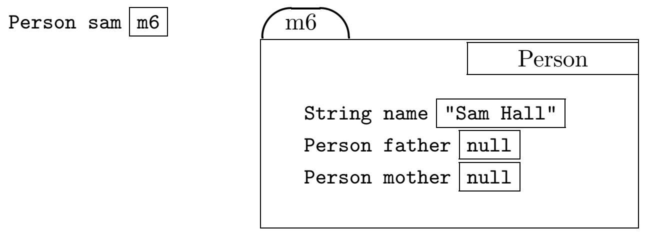
Question 4. The steps in evaluating a `new` expression **new** `C(...)` are:

1. Create a new object of the class and initalize its fields according to the declarations of the fields.
2. Execute the constructor call `C(...)`.
3. Yield (return) the name of the newly created object.

Question 5. If a class `C` does not contain a constructor, the constructor shown below is used; however, if `C` does contain a constructor, the one shown below is *not* automatically included.

public `C(){}`

Question 6. Below, we show the result of executing the assignment statement:



Activity 3-5-3 Calling a constructor from a constructor

Question 7. The form of a constructor call within a constructor is:

this(<*arguments*>);

Question 8. Below is the rewritten Pez example:

```
// An instance is (models) a Pez Dispenser
public class PezDispenser {
    String flavor;   // Flavor of this Pez container
    int numberOfPez; // Num candies in this container

    // Constructor: instance with flavor f and c candies
    public PezDispenser(String f, int t) {
        flavor= f;
        numberOfPez= t;
  }
```

```
        // Constructor: instance with flavor f and 20 candies
        public PezDispenser(String f) {
            this(f, 20);
        }

        // Test PezDispenser constructors.
        public static void main(String[] bob) {
            PezDispenser p1;
            p1= new PezDispenser("strawberry", 12);
            PezDispenser p2;
            p2= new PezDispenser("grape");

            System.out.println("Pez 1 information: " +
                p1.flavor + " " + p1.numberOfPez);
            System.out.println("Pez 2 information: " +
                p2.flavor + " " + p2.numberOfPez);
        }
    }
```

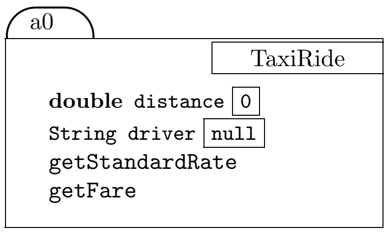
Lesson page 3-6. Nonstatic methods

Activity 3-6-1 Nonstatic methods

Question 1. A nonstatic method is a method that is defined in a class without prefix **static**; thus, it is an instance method.

Question 2. An instance method is a method that is defined in a class without prefix **static**. The method appears in each instance of the class.

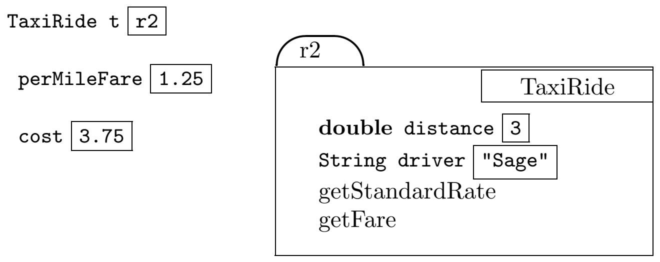
Question 3. The object looks like this:



Question 4. There is no need to explicitly initialize the field because it is automatically initialized to 0.

Question 5. The statement is: **double** `cost= t.getFare();` .

Question 6. Here is what the variables look like after execution:



Question 7. Here is the specification:

```
// Add 1 mile to the distance this Taxi has traveled.
```

Activity 3-6-2 Creating an instance of `Circle`

Activity 3-6-3 Using instance methods of class `Circle`

Activity 3-6-4 Hiding instance variables

Question 8. Keyword **private** is an access modifier. Its presence as a prefix of a class, method, or field definition indicates that the entity is visible (can be accessed) only in the class in which it is defined. It can't even be accessed in a subclass.

Question 9. A getter method is a method that returns some value from the instance in which the method appears —usually, but not always, the value of an instance variable.

Question 10. A setter method stores a value in (or sets) an instance variable of an object in which it appears.

Question 11. Here is the revied class:

```
public class TaxiRide {
    private double distance; // Miles traveled so far
    private String driver;   // Taxi driver's name

    // Constructor: an instance with driver d
    public TaxiRide(String d)
        { driver= d; }

    // = the per-mile fare for every taxi, in dollars
    public static double getStandardRate()
        { return 1.25; }

    // = the amount owed for this ride
    public double getFare()
        { return distance * getStandardRate(); }
    // = miles traveled so far during this taxi ride
    public double getDistance()
        { return distance; }

    // = this taxi driver's name
    public String getDriver()
        { return driver; }
}
```

Question 12. Here is method `setDistance`:

```
// Set the distance this taxi ride has covered to d.
public void setDistance(double d)
    { distance= d; }
```

Activity 3-6-5 Reasons for using modifier `private`

Question 13. Two reasons for making an instance variable private are:

1. **private** helps provide controlled access.
2. **private** makes some later changes easier.

Activity 3-6-6 Check your understanding of nonstatic methods

Lesson page 3-7. Consequences of using objects

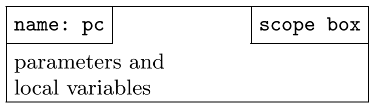
Activity 3-7-1 The scope box for a call of an instance method

Question 1. The scope box is used to determine where to look for names that appear in the method body by that are not in the frame for the method call.

Question 2. The scope box for an instance method contains the name of the object in which the called method resides.

Question 3. The scope box for a static method contains the name of the class in which the method is defined.

Question 4. In the frame below, `name` is the name of the method and `pc` is the program counter.



Question 5. Here are the steps in executing a function call:

1. Evaluate the arguments of the call;

2. Draw a frame for the call —don't forget to put in the scope box;.
3. Draw in the parameters;
4. Assign the values of the arguments to the parameters.
5. Execute the method body; stop when a return statement **return** `e;` is executed, and evaluate `e`;
6. Erase the frame —pop it from the call stack;
7. Push onto the call stack the value of `e`.

Question 6.

For one of these	The scope box contains
Constructor	Name of the newly created object
Static method	Name of the class
Non-static method	Name of the object in which the method resides

Question 7. Note that to compare the `drivers`, you have to use method `String.equals`. If you used the test **this.driver == t.driver**, the comparison would have been between the names of the `Strings`, and not the `Strings` themselves.

```
// = ``this instance and t are equal (have the
//      same driver and went the same distance)''
public boolean equals(TaxiRide t) {
    return this.driver.equals(t.driver) &&
        this.distance == t.distance;
}
```

Question 8. An alias is an assumed name, another name. In programming, aliasing refers to the use of two names for the same object.

Activity 3-7-2 Method `toString`

Question 9. Method `toString` is used to produce a `String` representation of an object.

Question 10. We have "rectangled" rather than "circled" the statements that implicitly call method `toString`.

`Employee e= new Employee();`

$\boxed{\texttt{System.out.println(e);}}$

`Employee f= e;`

$\boxed{\texttt{String s= "" + e;}}$

$\boxed{\texttt{System.out.println("" + e);}}$

$\boxed{\texttt{String t= e + "";}}$

Activity 3-7-3 Evaluating a call on method `toString`

Question 11. The output is:

```
Employee Petra, start 2000, salary 200000
Employee Paul, start 1997, salary 100000
```

Activity 3-7-4 Further examples of method `toString`

Question 12. Here's method `toString`:

```
// = String representation of this object.
public String toString() {
    return "The driver is " + driver +
        ", distance traveled is " + distance +
        ", and $" + getFare() + " is the fare.";
}
```

Activity 3-7-5 Exercises on objects

Lesson page 3-8. Object-oriented design

Activity 3-8-1 Designing the classes of a program

Question 1. The problem domain is the body of knowledge for the area for which the program is being written or that the program is supposed to model.

Question 2. The specification of a class defines all non-private entities of the class —basically, the public methods and variables.

Question 3. The basic steps of object-oriented design are listed below.

1. Write noun phrases to describe the objects to be manipulated.
2. Decide on classes and write their specifications.
3. Implement the classes.
4. Put it all together and test it.

Activity 3-8-2 Problem statement: reading a clock

Question 4. The information requested in the first two windows are (1) the name of the player and (2) the level they would like to start at.

Question 5. Here are the four levels of play:

1. The player is asked only for hours, and the clock always shows 0 minutes.
2. The clock shows 0, 15, 30, or 45 minutes.
3. The clock shows minutes that are a multiple of 5.

4. The clock shows any minute in the range 0..59.

Question 6. The level is incremented whenever the score reaches a multiple of 5 and the level is still less than 4.

Activity 3-8-3 Identifying the classes

Question 7. We can't answer this one for you.

Activity 3-8-4 Designing classes `Clock` and `Time`

Question 8. Here are the specifications of classes `Clock` and `Time`:

```
// A Clock is a window, with a clock face and time
public class Clock extends Canvas {
    // Constructor: a clock with time t
    public Clock(Time t)

    // = the time on the clock
    public Time getTime()

    // Set the clock time to t
    public void setTime(Time t)

    // Paint the clock using g
    public void paint(Graphics g)
}

// A time, in minutes (in 0..59) and hours (in 0..11)
public class Time {
    // Constructor: an instance with h hours and m minutes
    public Time(int h, int m)

    // = representation ``hours:minutes'' of this Time
    public String toString()

    // = ``this Time equals Time t''
    public boolean equals(Time t)
}
```

Activity 3-8-5 Implementing `ClockWindow`, `Clock`, and `Time`

Activity 3-8-6 Designing the player and the game

Question 9. Here are the specifications of classes `Player` and `ClockGame`:

```
// An instance is a player playing the clock game
public class Player {
    // Constructor: a player with name s,
    // score 0, and playing level lev
    public Player(String s, int lev)

    // Get the players name, score, and level
    public String getName()
    public int getScore()
    public int getLevel()

    // Increment the score (and level if necessary)
    public void incrementScore()

    // a String representation of the player
    public String toString()
}

// A game
public class ClockGame {
    // Constructor: a player with fields initialized
    // and a clock window with a clock and a label
    public ClockGame()

    // Play until the player terminates the game
    public void playGame()
}
```

Activity 3-8-7 Implementing the player and the game

Activity 3-8-8 Putting it all together

Question 10. We provide no answer to this exercise.