- When should an object be deleted? And if it is going to be deleted, why not just leave it out of the code entirely?
  - some languages require the programmer to explicitly deallocate objects once they are no longer needed
  - this is not necessary (or even possible) in Java
    - Java's garbage collector automatically cleans up and deallocates objects that are no longer accessible to the program
  - many things may only be needed temporarily – that doesn't mean they shouldn't ever exist

`if` and `{}`

- you can omit `{}` in `if` statements if there is only a single statement in a part, but it is recommended that you *don't* do this because of the potential for bugs

```
if ( x > 0 )
  System.out.println(x);
  x++;
```

  - this looks like both statements happen only if x > 0, but remember that program semantics do not depend on the indentation – actually only the `System.out.println` is inside the `if`

Classes and Objects

- a *type* involves a set of legal values and the operations that can be applied to those values
- a *class* provides a definition for a user-defined type
- an *object* is a particular instance of a class
- an object is a black box which contains some state (values), with certain ways to access or manipulate that state
- objects in a program are used to represent real-world objects
  - the object's state represents the real object's properties
  - the object's operations manipulate its state in the way that you interact with the real world object and manipulate its properties
- a class defines an object's properties and operations

Imagine that you were designing a class Dice to represent a single die that you could use in a game like Pig. For each thing listed, identify whether it is a property of a die, an operation of a die, or neither.

roll the die	[Choose]
operation (roll)	
check if the player rolled a 1	[Choose]
operation (get value rolled)	
the player's current score	[Choose]
the number of sides the die has	[Choose]
property (number of sides)	
the player's total for the current turn	[Choose]
the number currently on top of the die	[Choose]
property (current value)	

properties are values stored – instance variables

functions are operations that can be applied – methods

Writing Classes

In Java, a class generally has one of two purposes –

- a holder of subroutines (such as `main`)
  - all elements (subroutines, global variables, global constants) are `static`
- a blueprint for creating objects
  - most elements are not `static` (exception is global constants)

Writing Classes

Elements of a class used to define objects –

- instance variables
  - these define the object's state – values that can be different for different objects and/or different at different times for one object
- one or more constructors
  - to initialize the instance variables
- methods
  - these define the operations that can be used to access and manipulate the object's state
  - may include getters and setters

Writing Classes – Syntax

- each public class goes in its own file, with the class name matching the file name

```
/**
 * Describe the purpose of the class.  (What
 * kind of thing does this class describe?)
 *
 * @author author's name
 */
public class ClassName {
  …
}
```

  - convention is to start class names with a capital letter (to distinguish from primitive types)

Writing Classes – Syntax

- instance variables define the object's state

```
public class ClassName {
  private type varname_; // description
  …
}
```

  - typically `private` rather than `public` (for encapsulation and information hiding)
  - not `static`
  - naming conventions
    - start with lowercase letter
    - end with _ to distinguish from local variables and parameters (note: this convention is not used in the book)
  - in some cases can be initialized at the point of declaration but more typically initialized in the constructor

Instance variables should generally be

- ○ public
- ★ private
- ○ public or private, either is fine
- ○ neither public nor private

Where are instance variables initialized? Choose all that apply.

- ☆ when they are declared
- ☐ by the caller of the constructor
- ★ in the constructor
- ☐ in a getter
- ☐ in a setter

Writing Classes – Syntax

- constructors create new objects
  - responsible for any setup that is required before an object can be used – typically initializing instance variables

```
public class ClassName {
  /**
   * Description.
   */
  public ClassName ( param-list ) {
    …
  }
}
```

also include @param tags for each parameter

  - not `static`
  - no return type or value (not even `void`)
  - constructor name must match the class name
  - can have any number of parameters (*default constructor* if 0)
  - can have multiple constructors but it must be possible to distinguish them by the number and/or type of their parameters

Writing Classes – Syntax

- methods implement operations
  - access and/or manipulate object's state

```
public class ClassName {
  /**
   * Description.
   */
  public return-type name ( param-list ) {
    …
  }
}
```

  - `public` methods are intended for use outside the class
  - `private` helper methods support the implementation of other methods but are not available outside the class
  - not `static`
  - naming conventions – generally same as subroutines/functions
    - getters – `get`*`Something`* (`is`*`Something`* for `boolean` return values)
    - setters – `set`*`Something`*