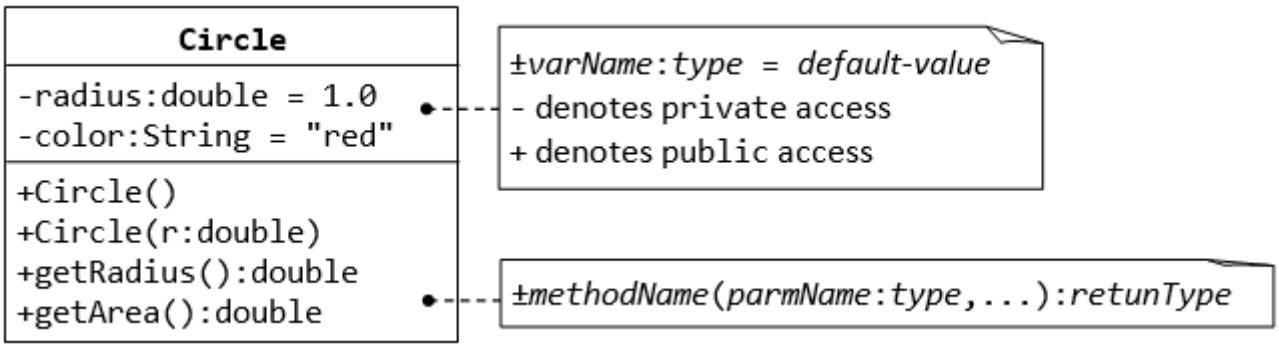
1. Exercises on Classes

Jednoduchá cvičení na třídy. Připomínám, že psáno pro Javu (proto je zde printLn)

1.1 Ex: The `Circle` Class (An Introduction to Classes and Instances)

This first exercise shall lead you through all the *basic concepts* in OOP.

viz: Diagram tříd na Wikipedii



A class called **`circle`** is designed as shown in the following class diagram. It contains:

- Two `private` instance variables: `radius` (of the type `double`) and `color` (of the type `String`), with default value of `1.0` and "`red`", respectively.
- Two *overloaded* constructors - a *default* constructor with no argument, and a constructor which takes a double argument for radius.
- Two `public` methods: `getRadius()` and `getArea()`, which return the radius and area of this instance, respectively.

The source codes for `Circle.java` is as follows:

```
/*
 * The Circle class models a circle with a radius and color.
 */
public class Circle {  // Save as "Circle.java"
   // private instance variable, not accessible from outside this class
   private double radius;
   private String color;

   // The default constructor with no argument.
   // It sets the radius and color to their default value.
   public Circle() {
      radius = 1.0;
      color = "red";
   }

   // 2nd constructor with given radius, but color default
   public Circle(double r) {
      radius = r;
      color = "red";
   }

   // A public method for retrieving the radius
   public double getRadius() {
     return radius;
```

```
   }

   // A public method for computing the area of circle
   public double getArea() {
      return radius*radius*Math.PI;
   }
}
```

Compile "`Circle.java`". Can you run the `Circle` class? Why?

This `Circle` class does not have a `main()` method. Hence, it cannot be run directly. This `Circle` class is a "building block" and is meant to be used in another program.

Let us write a *test program* called `TestCircle` (in another source file called `TestCircle.java`) which uses the `Circle` class, as follows:

```
public class TestCircle {  // Save as "TestCircle.java"
   public static void main(String[] args) {
      // Declare an instance of Circle class called c1.
      // Construct the instance c1 by invoking the "default" constructor
      // which sets its radius and color to their default value.
      Circle c1 = new Circle();
      // Invoke public methods on instance c1, via dot operator.
      System.out.println("The circle has radius of "
         + c1.getRadius() + " and area of " + c1.getArea());

      // Declare an instance of class circle called c2.
      // Construct the instance c2 by invoking the second constructor
      // with the given radius and default color.
      Circle c2 = new Circle(2.0);
      // Invoke public methods on instance c2, via dot operator.
      System.out.println("The circle has radius of "
         + c2.getRadius() + " and area of " + c2.getArea());
   }
}
```

Now, run the `TestCircle` and study the results.

More Basic OOP Concepts

1. **Constructor:** Modify the class `Circle` to include a third constructor for constructing a `Circle` instance with two arguments - a `double` for `radius` and a `String` for `color`.

```
// 3rd constructor to construct a new instance of Circle with the given radius and color
```

```
public Circle (double r, String c) { ..... }
```

   Modify the test program `TestCircle` to construct an instance of `Circle` using this constructor.

Getter: Add a getter for variable `color` for retrieving the `color` of this instance.

```
// Getter for instance variable color
```

```
public String getColor() { ..... }
```

   Modify the test program to test this method.

`public` vs. `private`: In `TestCircle`, can you access the instance variable `radius` directly (e.g., `System.out.println(c1.radius)`); or assign a new value to `radius` (e.g., `c1.radius=5.0`)? Try it out and explain the error messages.

Setter: Is there a need to change the values of `radius` and `color` of a `Circle` instance after it is constructed? If so, add two `public` methods called *setters* for changing the `radius` and `color` of a `Circle` instance as follows:

```
// Setter for instance variable radius
public void setRadius(double newRadius) {
   radius = newRadius;
}

// Setter for instance variable color
public void setColor(String newColor) { ..... }
```

Modify the `TestCircle` to test these methods, e.g.,

```
Circle c4 = new Circle();   // construct an instance of Circle
c4.setRadius(5.0);          // change radius
System.out.println("radius is: " + c4.getRadius()); // Print radius via getter
c4.setColor(......);        // Change color
System.out.println("color is: " + c4.getColor());   // Print color via getter

// You cannot do the following because setRadius() returns void,
// which cannot be printed.
System.out.println(c4.setRadius(4.0));
```

Keyword "this": Instead of using variable names such as `r` (for `radius`) and `c` (for `color`) in the methods' arguments, it is better to use variable names `radius` (for `radius`) and `color` (for `color`) and use the special keyword "`this`" to resolve the conflict between instance variables and methods' arguments. For example,

```
// Instance variable
private double radius;

// Constructor
public Circle(double radius) {
   this.radius = radius;   // "this.radius" refers to the instance variable
                           // "radius" refers to the method's parameter
   color = .......
}

// Setter of radius
public void setRadius(double radius) {
   this.radius = radius;   // "this.radius" refers to the instance variable
                           // "radius" refers to the method's argument
}
```

Modify ALL the constructors and setters in the `Circle` class to use the keyword "`this`".

Method `toString()`: Every well-designed Java class should contain a `public` method called `toString()` that returns a short description of the instance (in a return type of `String`). The `toString()` method can be called explicitly (via *`instanceName`*`.toString()`) just like any other method; or implicitly through `println()`. If an instance is passed to the `println(`*`anInstance`*`)` method, the `toString()` method of that instance will be invoked implicitly. For example, include the following `toString()` methods to the `Circle` class:

```
// Return a description of this instance in the form of
```

```
// Circle[radius=r,color=c]
public String toString() {
   return "Circle[radius=" + radius + " color=" + color + "]";
}
```

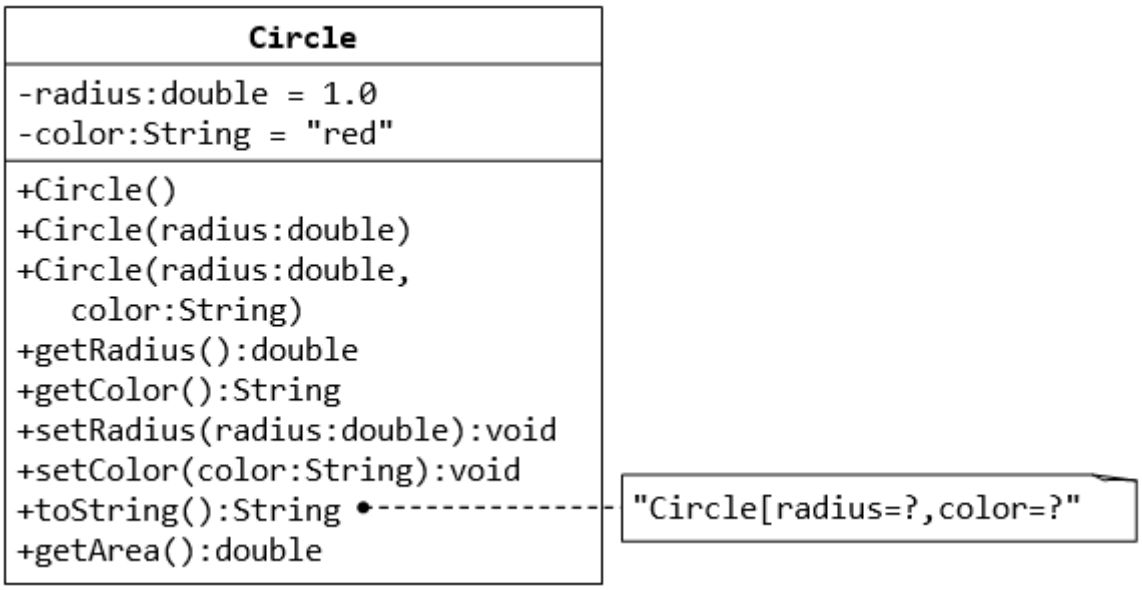
Try calling `toString()` method explicitly, just like any other method:

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

System.out.println(c1.toString());   // explicit call
```

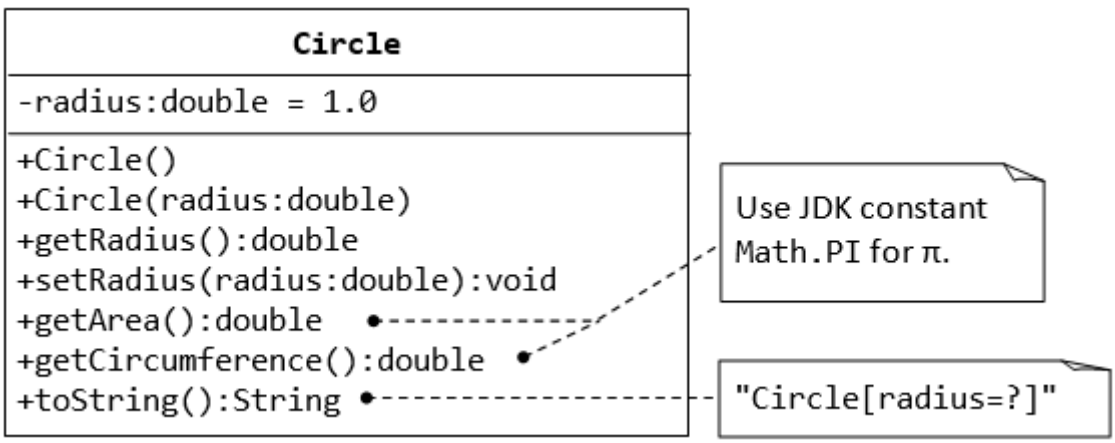
`toString()` is called implicitly when an instance is passed to `println()` method, for example,

```
Circle c2 = new Circle(1.2);

System.out.println(c2.toString());  // explicit call
System.out.println(c2);             // println() calls toString() implicitly, same as above
System.out.println("Operator '+' invokes toString() too: " + c2);   // '+' invokes toString() too
```

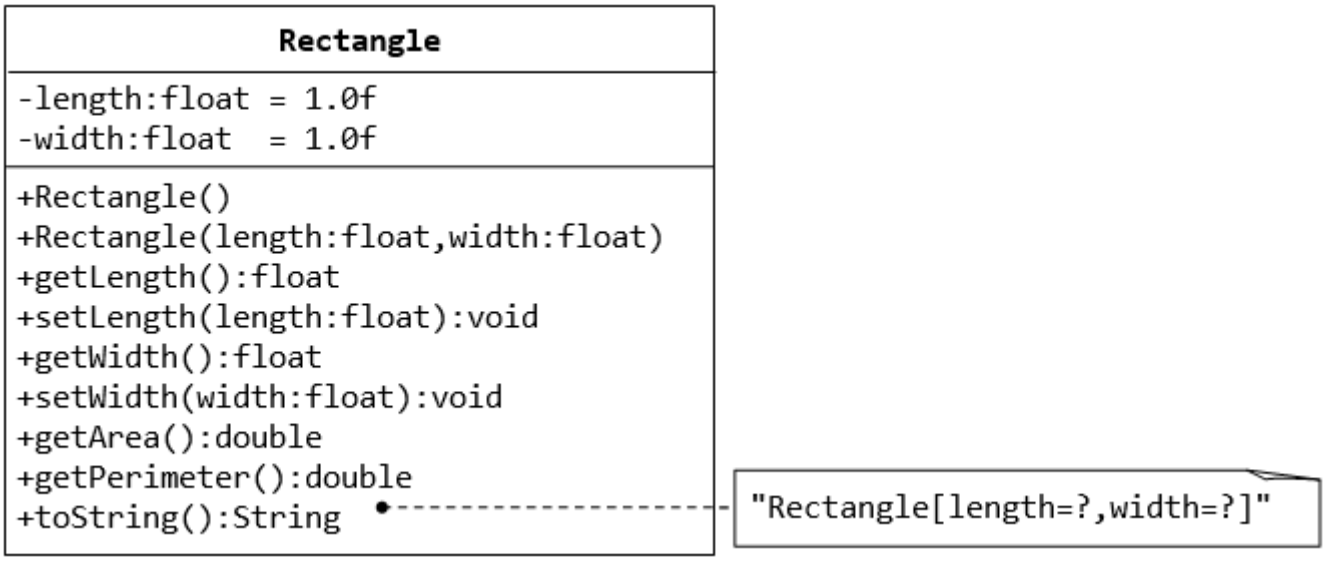
The final class diagram for the `Circle` class is as follows:



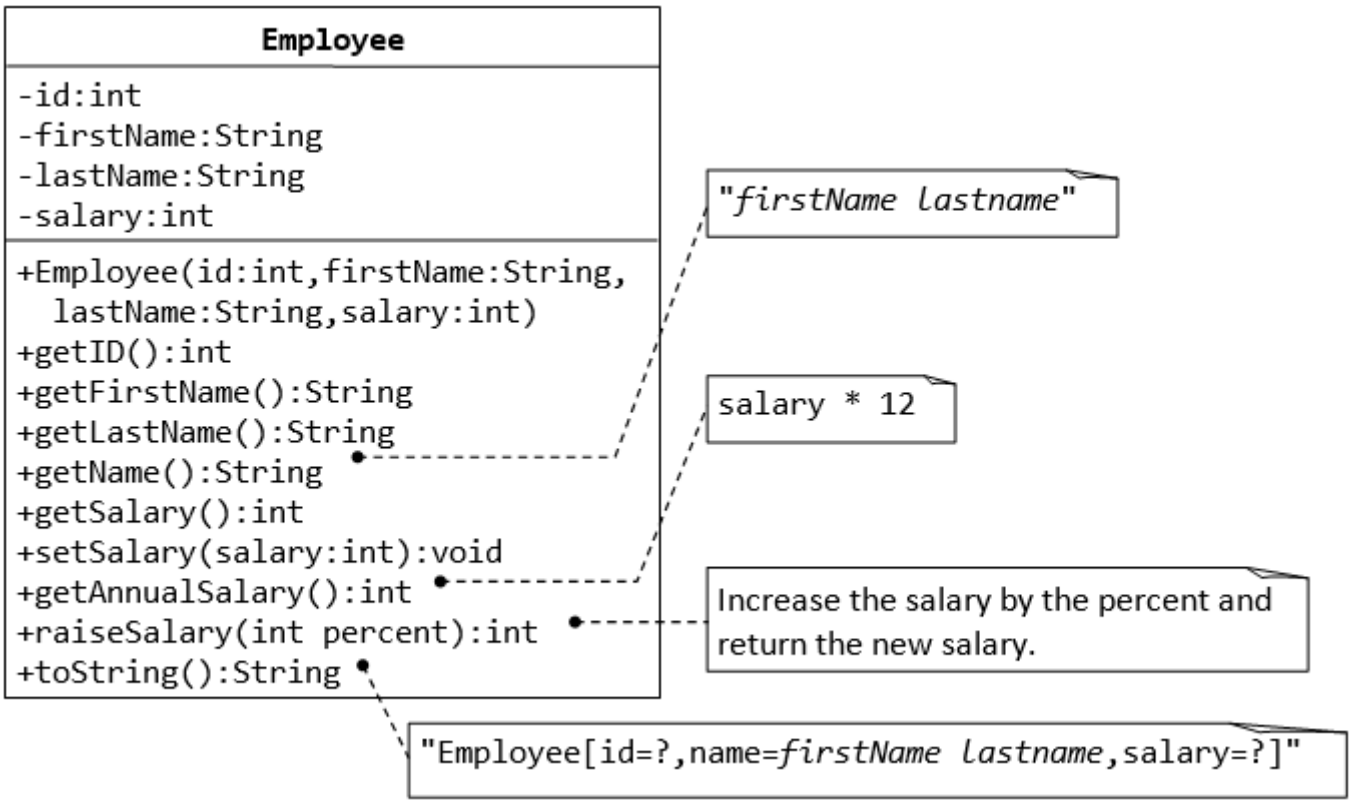
1.2 Ex: A Simplified `Circle` Class



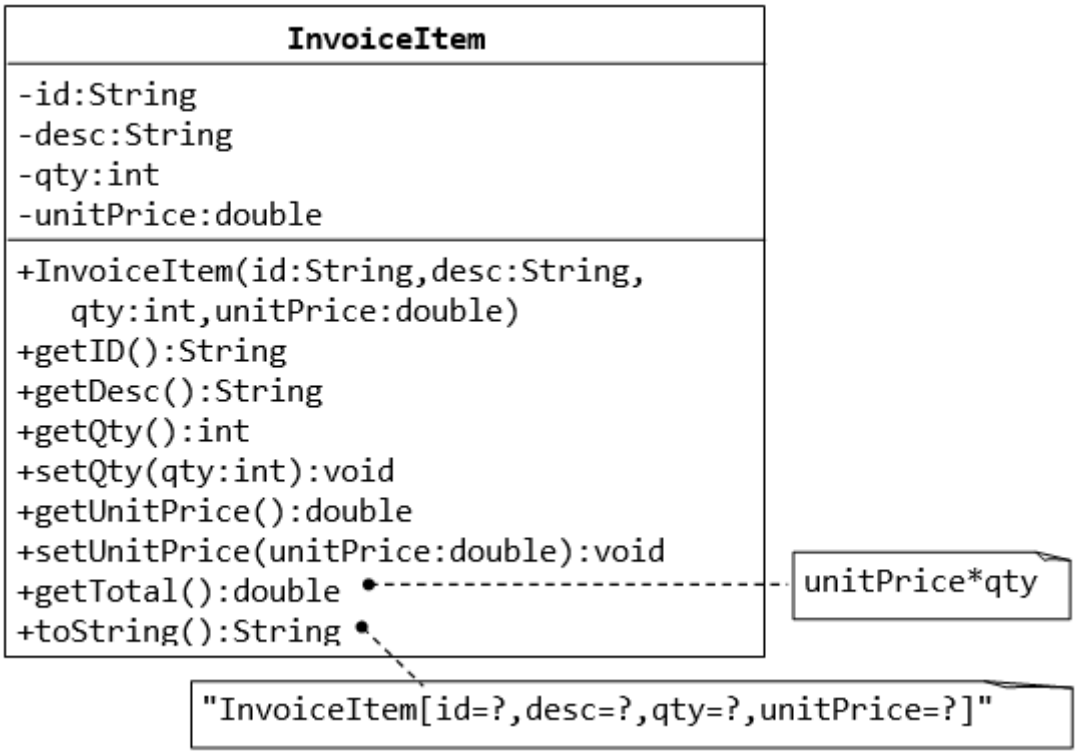
1.3 Ex: The `Rectangle` Class



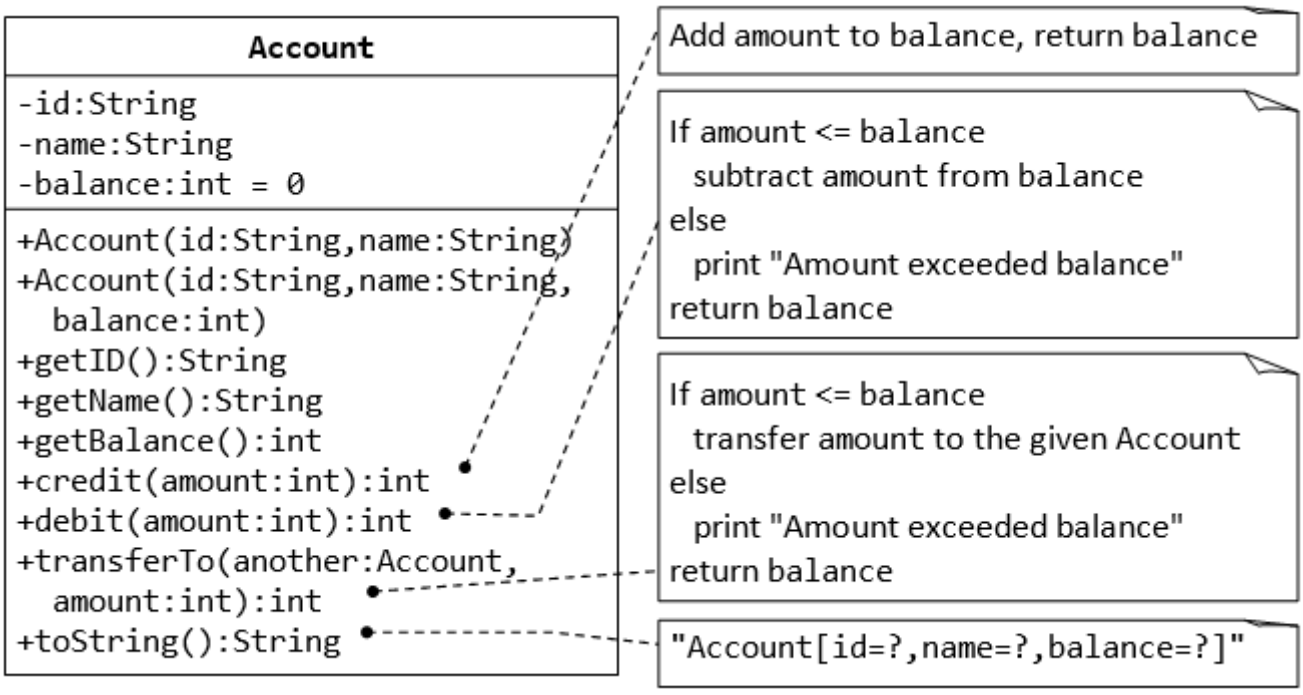
1.4 Ex: The `Employee` Class



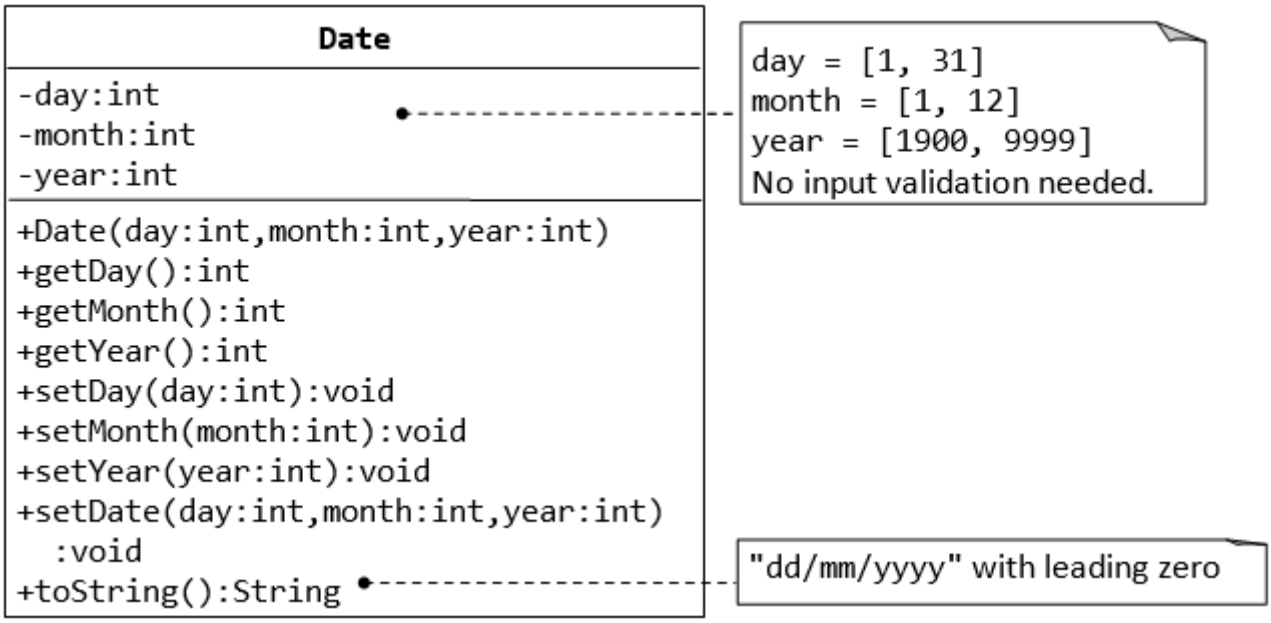
1.5 Ex: The `InvoiceItem` Class



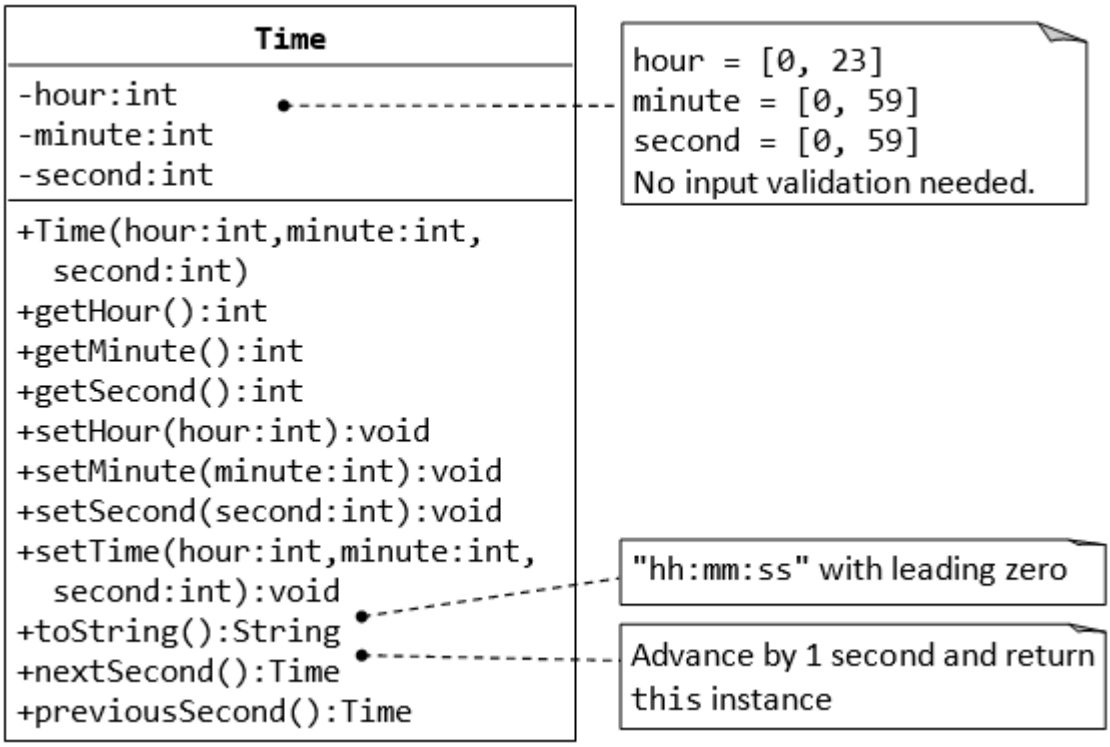
1.6 Ex: The Account Class



1.7 Ex: The Date Class



1.8 Ex: The `Time` Class



2