Chapter 1 Exercise

The exercises in this chapter help practise class design with UML class diagram notation and design specification. To ease reading, some of the relevant exercises given in [19] are reproduced here.

1. (**Greeting Conversation**) Figure 1.17 is an extended design diagram of the greeting conversation program discussed in the previous tutorial. This diagram shows a few more attributes of the two classes `Person` and `MobilePhone`, but it presents only a partial list of operations.

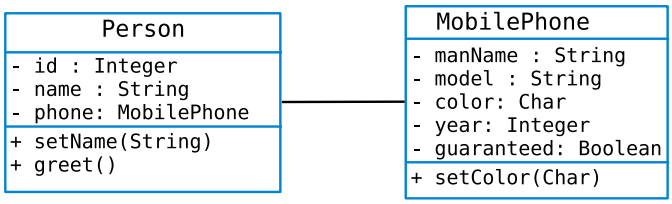

Figure 1.17: An initial design diagram for the greeting conversation program.

The following table is a partially completed table of the domain constraints that apply to the attributes of the two classes in Figure 1.17. Note that the "type" column lists the formal types of the attributes. Attribute `MobilePhone.color` takes a character value that denotes a colour of the phones. The characters are: 'R' (for red), 'O' (for orange), 'Y' (for yellow), 'B' (for blue), 'P' (for purple). Attribute `MobilePhone.guaranteed` takes the value `true` for a mobile phone if this phone has a guarantee; it takes the value `false` if otherwise.

Class	Attribute	type	mutable	optional	length	min	max
Person	id	Integer	F	F	-	1	-
	name	String	T	F	30	-	-
	phone	MobilePhone	T	T	-	-	-
MobilePhone	manName	String	F	F	100	-	-
	model	String	F	F	50	-	-
	color	Character	T	F	1	-	-
	year	Integer	F	F	-	1970	-
	guaranteed	Boolean	T	T	-	-	-

(a). Complete the domain constraints in the table, using your practical understanding of the application. ✓

(b). Write the initial design specification for each class, which must include the object representation.

(c). Determine a minimum set of operations needed for each class. Justify your choice of each operation.

(d). Update the design diagram with the operations that you identified in the previous task.

(e). Update the design specification of each class to include the operational specification.

2. (**Greeting conversation v1.1**) Implement an enum named `Color` that catures the different colours mentioned in the program requirement. Update class `MobilePhone` to use this enum.
3. (**Greeting conversation v1.2**) Update class `MobilePhone` to address an additional constraint that attribute `model` must be of the form `M-`$ABC-MNP$, where ABC is a 3-letter word and MNP is a 3-digit word. For example, `M-SAM-123` is a valid phone model, but `M-SOM-123` is not.
4. (**Greeting conversation v1.3**) Update class `Person` to address an additional constraint that attribute `name` must consist of at least two words that are separated by a white space.
5. Specify a class `EvenIntSet` that represents a set of even numbers. This is an integer set that only accepts even numbers as elements.
6. (see [19]) Specify a map class, named `StringIntMap`, which maps strings to integers. Maps allow an existing mapping to be looked up. Maps are also mutable: new pairs can be added to a map, and an existing mapping can be removed. Be sure that your data type is adequate.
7. (see [19]) Specify a class `IntQueue` that represents a bounded queue of integers. A bounded queue is a queue that has an upper bound, established when the queue is created, on the number of integers that can be stored in the queue. Queues are mutable and provide access to their elements in first-in/first-out order. `IntQueue` operations include:

   ```
   IntQueue(int n)
   enq(int x)
   int deq()
   ```

   The constructor creates a new queue with maximum size `n`, `enq` adds an element to the front of the queue, and `deq` removes the element from the end of the queue. You may include extra operations as needed for adequacy.
8. (see [19]) Specify a rational number type, named `Rat`.

Chapter 2 Exercise

The exercises in this chapter continue from those in Chapter 1 with two main objectives: (1) review the design and (2) implement the design in Java. Note that some exercises were reproduced from [19].

1. (**Greeting Conversation**) Review the design of and implement the `GreetingConversation` program of Exercise 1.1:
   (a). Review the design of and implement class `Person`.
   (b). Review the design of and implement class `MobilePhone`.
   (c). Create an program class named `GreetingConversation`, whose `main` method performs the following basic object manipulation tasks:
   - create a `MobilePhone` object.
   - create a `Person` object.
   - check that the objects are valid and if so display information about them, else display suitable error messages.
2. (**Greeting Conversation v1.1–1.3**) Review the design of and implement the three improvements to `GreetingConversation` discussed in Exercise 1.2-Exercise 1.4. Update method `GreetingConversation.main` to make use of the improved design.
3. Review the design of and implement class `EvenIntSet` of Exercise 1.5. Create an program class named `EvenIntegers`, whose `main` method performs the basic object manipulation of `EvenIntSet`.
4. Review the design of and implement class `StringIntMap` Exercise 1.6. Create an program class named `MixedMaps`, whose `main` method performs the basic object manipulation of `StringIntMap`.
5. Review the design of and implement class `IntQueue` Exercise 1.7. Create an program class named `NumQueue`, whose `main` method performs the basic object manipulation of `IntQueue`.
6. Review the design of and implement class `Rat` Exercise 1.8. Create an program class named `Rationals`, whose `main` method performs the basic object manipulation of `Rat`.