

CSCI 3155: Homework Assignment 10

Practice Only

Data Abstraction

Exercise 1: Skill 18.1. Write an integer first-in-first-out (FIFO) queue in SML. You should define a **structure** `Queue` that ascribes to the `QUEUE` signature below. Note that the type `t` is opaque (i.e., its definition is not given in the signature `QUEUE`). Your `Queue` structure only needs to support these operations: an `empty` queue, an `enqueue` operation that adds an integer to end of the queue, and an `dequeue` operation that removes the next integer.

```
signature QUEUE = sig
  type t
  val empty : t
  val enqueue : t * int -> t
  val dequeue : t -> int * t
end
```

Discuss why your implementation is an abstract data type (i.e., how is information hidden), and demonstrate that it is so with example client code.

Exercise 2: Skill 18.2. Convert your `Queue` ADT in the previous question to be generic so that it works on arguments of any type. Demonstrate how you will use your generic structure. Hint: you will need to modify the signature above.

Exercise 3: Skill 18.1. Implement an integer queue using data abstraction in either C++ or Java. It should support the same operations as your SML abstract data type, but it should use information hiding mechanisms in C++/Java. That is, pay attention to what parts of your implementation are public and what parts are private. Discuss how your implementation hides information, and demonstrate that it does with example client code.

Exercise 4: Skill 18.2. Convert your integer queue in the previous question to be generic so that it works on arguments of any type. Demonstrate how you will use your generic class.

Exercise 5: Skill 18.3. Compare and contrast your queue implementations in SML versus C++/Java.

Object-Oriented Concepts

Exercise 6: Skill 19.3. Consider the following class hierarchy in which all methods are virtual (which is the default in Java):

```
class A {
    void m() { print "A's m implementation"; }
}
class B extends A {
    void m() { print "B's m implementation"; }
    void n() { print "B's n implementation"; }
}
class C extends B {
    void n() { print "C's n implementation"; }
}
```

In other words, C inherits from B and B inherits from A. Also B overrides A's m method and C overrides B's n method. What will be the output of the following code fragment. Explain your answer.

```
/* fragment 1 */
A a = new B();
a.m();
```

```
/* fragment 2 */
A a = new B();
a.n();
```