3.3 Abstract Data Type (ADT)

一级标题，二级标题，三级标题，四级标题，注，拓展，不懂

MIT 6.031：10、11

Safe from bugs	Easy to understand	Ready for change
Correct today and correct in the unknown future.	Communicating clearly with future programmers, including future you.	Designed to accommodate change without rewriting.

目录

3.3.1 Abstract Data Type

Definition of ADT（抽象数据类型的定义）

抽象：将一些事物的共同特点提取出来

抽象数据类型:将一些数据类型的共同特点(也就是对于该数据类型的操作)提取出来，而后，以这种共同操作定义一个具体实现的数据类型。

抽象数据类型（Abstract Data Type，ADT）是计算机科学中具有类似行为的特定类别的数据结构的数学模型。

抽象数据类型是间接定义的,通过其上的可执行的操作以及这些操作的效果的数学约束（与可能的代价）。

Characteristic：

A. 模糊了操作的具体实现形式，对于 client 而言，使之完全不可见，只可见其上的定义的操作
B. 一个抽象数据类型由其上的操作完全定义

Why use it（它是其中的思想的实例）

Abstract data types are an instance of a general principle as following in software engineering

A. Abstraction.（抽象） Omitting or hiding low-level details with a simpler, higher-level idea.
B. Modularity. （模块化）Dividing a system into components or modules, each of which can be designed, implemented, tested, reasoned about, and reused separately from the rest of the system.
C. Encapsulation. （封装）Building a wall around a module so that the module is responsible for its own internal behavior, and bugs in other parts of the system can't damage its integrity.
D. Information hiding.（信息隐藏） Hiding details of a module's implementation from the rest of the system, so that those details can be changed later without changing the rest of the system.
E. Separation of concerns.（功能分离）Making a feature (or "concern") the responsibility of a single module, rather than spreading it across multiple modules.

拓展：Classifying operations

Creators （构造器：从无到有）

definition

Creators create new objects of the type.

A creator may take values of other types as arguments, but not an object of the type being constructed.

implement

A creator is either implemented as a constructor , like new ArrayList(), or simply a static method instead, like Arrays.asList(), List.of(). 构造器：可能实现为构造函数或静态函数

拓展：a factory method

A creator implemented as a static method is often called a factory method 实现为静态方法的构造器通常称为工厂方法

Producers（生产器：从旧到新）

Producers create new objects of the type from one or more existing objects of the type.

For example , The concat（） method of String is a producer: it takes two strings and produces a new string representing their concatenation.

Observers（观察器：从数据中读取信息）

Observers take objects of the abstract type and return objects of a different type. The size method of List, for example, returns an int.

Mutators（变值器：改变数据的组成）

Mutators change objects. The add method of List, for example, mutates a list by adding an element to the end.

注意：这四种分类并不是严格的分类，每一种操作只能非此即彼，可以存在某种操作同时占据两个种类。

拓展：对于这四种分类的非正式的表示

A. creator : t* → T
B. producer : T+, t* → T
C. observer : T+, t* → t
D. mutator : T+, t* → void | t | T

其中：

a) Each T is the abstract type itself; each t is some other type.
b) The + marker indicates that the type may occur one or more times in that part of the signature.
c) The * marker indicates that it occurs zero or more times.
d) | indicates or.

3.3.2 Goals of an abstract type

choose good operations and determine how they should behave.

原则 1：设计简洁，一致的操作

It's better to have a few, simple operations that can be combined in powerful ways, rather than lots of complex operations.简洁

Each operation should have a well-defined purpose, and should have a coherent behavior rather than a panoply of special cases.内聚度高

We probably shouldn't add a sum operation to List, for example. It might help clients who work with lists of integers, but what about lists of strings? Or nested lists? All these special cases would make sum a hard operation to understand and use.一致：保证所有的具体实现形式都可以完成。

原则 2：完备，且有效

The set of operations should be adequate in the sense that there must be enough to do the kinds of computations clients are likely to want to do.（要足以支持 client 对数据所做的所有操作需要，且用操作满足 client 需要的难度要低）

A good test is to check that every property of an object of the type can be extracted. 判断方法：对象每个需要被访问到的属性是否都能够被访问到。

For example, if there were no get operation, we would not be able to find out what the elements of a list are. 没有 get()操作就无法获取 list 的内部数据。

Basic information should not be inordinately difficult to obtain.必须有效的获取基本数据信息。

For example, the size method is not strictly necessary for List, because we could apply get on increasing indices until we get a failure, but this is inefficient and inconvenient. 用遍历方式获取 list 的 size –太复杂 vs 提供 size()操作方便 client 使用。

原则 3：通用，特殊二选一

The type may be generic: a list or a set, or a graph, for example. Or it may be domain-specific: a street map, an employee database, a phone book, etc.

But it should not mix generic and domain-specific features.

For example, A Deck type intended to represent a sequence of playing cards shouldn't have a generic add method that accepts arbitrary objects like integers or strings. Conversely, it wouldn't make sense to put a domain-specific method like dealCards into the generic type List.

面向特殊应用的类型不应包含通用方法。

面向通用的类型不应包含面向特殊应用的方法。

3.3.3 Representation independence（表示独立性，ADT 最重要性质）

Definition：

the use of an abstract type is independent of its representation.

client 使用 ADT 时无需考虑其内部如何实现，ADT 内部表示的变化不应影响外部 spec

和客户端。

Roles

its operations are fully specified with preconditions and postconditions, so that clients know what to depend on, and you know what you can safely change.是一个 ADT 的必要性质。

拓展：connection to our three key properties of good software

Safe from bugs.

A good ADT offers a well-defined contract for a data type, so that clients know what to expect from the data type, and implementers have well-defined freedom to vary.

Easy to understand.

A good ADT hides its implementation behind a set of simple operations, so that programmers using the ADT only need to understand the operations, not the details of the implementation.

Ready for change.

Representation independence allows the implementation of an abstract data type to change without requiring changes from its clients.

3.3.4 Invariants（不变量）

Definition

An invariant is a property of a program that is always true, for every possible runtime state of the program. （不变量：程序在任何时候总是 true 的性质）

Characteristic

The most important property of a good abstract data type is that it preserves its own invariants. ADT 需要始终保持其不变量是一个好的 ADT 的最重要的性质，由于 ADT 的一个特性（表示独立性）就是将客户端与实施者进行隔离，不变量的存在会实现这种隔离，实现表示独立性。

When an ADT preserves its own invariants, reasoning about the code becomes much easier.你可以排除可能由不变量引起的错误的可能性。

Implement

在参数与返回值处使用防御式拷贝，但是，最好的方案还是使用不变的数据类型。

注：在某种程度上而言，不变量的定义没有那么重要，完全可以建立起从表示不变量到表示独立性的关系

正是因为表示不变量（也就是对于该 ADT 有用的值）的保持，表示不变量是一个 ADT 实现其表象的具体实现对象，保证表示不变量不变就使得 client 无法修改一个 ADT 实现，实现了 client 和 implement 的隔离。

3.3.5 abstraction function and rep invariant（抽象函数与表示不变量）

The Space of representation values and abstract values

The space of representation values（表示空间）

consists of the values of the actual implementation entities（实现者看到和使用的值，具体而言，表示空间由实现者选用的类型的所有值组成）

In simple cases, an abstract type will be implemented as a single Java object, but more commonly a small network of objects is needed.（在简单的情况下，一个抽象类型只需要实现为单个的对象，但是更常见的情况是使用一个很多对象的网络）

The space of abstract values（抽象空间）

consists of the values that the type is designed to support.（client 看到和使用的值）

该空间中的值都是虚假的，不是真实存在，但这些值也是客户端所需要的。

For example, an abstract type for unbounded integers, like Java’s BigInteger, might have the mathematical integers as its abstract value space; the fact that it might be implemented as an array of primitive (bounded) integers, say, is not relevant to the user of the type.

Relationship between them

the implementer’s job to achieve the illusion of the abstract value space using the rep value space.

A. Every abstract value is mapped to by some rep value

每一个在抽象空间的值一定在表示空间会有对应。

The purpose of implementing the abstract type is to support operations on abstract values. Presumably, then, we will need to be able to create and manipulate all possible abstract values, and they must therefore be representable.

B. Some abstract values are mapped to by more than one rep value

某些在抽象空间的值在表示空间可能存在多个对应。

This happens because the representation isn't a tight encoding. There's more than one way to represent an unordered set of characters as a string.由于表示形式并非严格的编码，故而对于在抽象空间的值可能有多种实现。

C. Not all rep values are mapped

可能存在没有抽象量对应的表示量。

In this case, the string “abbc” is not mapped, because we have decided that the rep string should not contain duplicates. This will allow us to terminate the remove method when we hit the first instance of a particular character, since we know there can be at most one.

注：需要明白的是，表示空间中的值为实现者所选类型的所有值的集合，只有这样，一切的关系疑惑迎刃而解。

abstraction function

definition

An abstraction function that maps rep values to the abstract values they represent

是一个映射：从（表示空间 / 无对应的值的集合）到（抽象空间）的映射，阐明有抽象值对应的表示值与抽象值的关系

Characteristic

必须是满射，可能是单射。

Determination

A common confusion about abstraction functions and rep invariants is that they are determined by the choice of rep and abstract value spaces, or even by the abstract value space alone.

在我看来，这是显而易见的，一个抽象函数说白了就是抽象空间与表示空间的笛卡尔积的子集，当然不可能有这两个集单独确定，更不要说被抽象集单独确定。

不过还是举个例子来进行说明。

A set of characters could equally be represented as a string, as above, or as a bit vector, with one bit for each possible character.

rep invariant

definition

A rep invariant that maps rep values to booleans。

是一个映射：从（表示空间）到（{flase，true}）的映射，阐明表示空间内的表示值是否存在有抽象值的对应。

其目的就是：确定 abstraction function 的定义域。

Characteristic

- 表示不变性 RI：某个具体的“表示”是否是“合法的”
- 也可将 RI 看作：所有表示值的一个子集，包含了所有合法的表示值
- 也可将 RI 看作：一个条件，描述了什么是“合法”的表示值

3.3.6 use assertion Checking the rep invariant

A. 可以使用断言来进行检查 RI 是否满足，来尽早地发现 bug。
B. 尽量在任何操作中都是用断言。
注：使用断言的原因是保证程序在发布时，可以避免判断的影响。
Ex：

```
// Check that the rep invariant is true
// *** Warning: this does nothing unless you turn on assertion checking
//     by running Java with -enableassertions (or -ea)
private void checkRep() {
    assert denominator > 0;
    assert gcd(Math.abs(numerator), denominator) == 1;
}
```

扩展：No null values in the rep

一般情况下，默认表示不变量不允许出现 null 值（包括数组或者列表中的元素），故而在表示不变量的定义中，没有必要出现其不为 null。

但是，当你对表示不变量进行检查时，必须检查该值不为 null 值，在大多数情况下，这时没有必要写的，因为其他调用该不变量的函数会抛出 null 异常。

当然也存在特殊情况，表示不变量可以为 null，，这是需要在表示不变量的声明中显式的定义该 RI 不为 null。

拓展：Beneficent mutation（友善改动）

the implementation is free to mutate a rep value as long as it continues to map to the same abstract value, so that the change is invisible to the client.

简单而言，在 ADT 中，实现者可以更改表示量，但保证抽象量的不变性，以达到对于客户端而言，这个 ADT 没有任何变化，这就是 Beneficent mutation（友善改动）。

这种友善改动可以使得程序的效率提高。

3.3.7 How to establish invariants（如何建立一个不变量）

To make an invariant hold, we need to:

A. make the invariant true in the initial state of the object; （确保在对象创建的时候不变量成立）

B. and ensure that all changes to the object keep the invariant true.（确保对对象在接下来的每一个改变后不变量依然成立）

Translating this in terms of the types of ADT operations, this means:

A. creators and producers must establish the invariant for new object instances; （创建者和生产者必须对新的对象就建立不变量）

B. and mutators and observers must preserve the invariant.（改造者和观察者必须保持/保护这种不变量）

PS：如此的和保真性或数学归纳法类似。

The risk of rep exposure makes the situation more complicated. If the rep is exposed, then the object might be changed anywhere in the program, not just in the ADT's operations, and we can't guarantee that the invariant still holds after those arbitrary changes. 表示暴露会使得情况更加复杂，如果一个表示被暴露出来，那么程序的任何地方都可能对其进行修改，我们也就没法确保不变量一直成立了。

总的来说，If an invariant of an abstract data type is

A. established by creators;

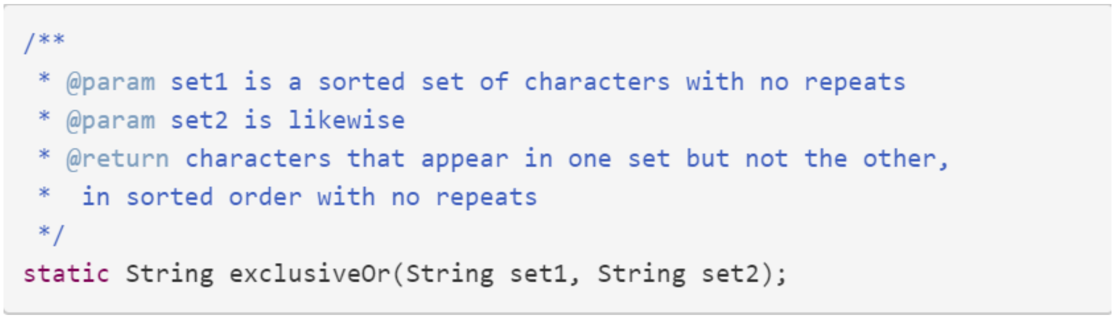
B. preserved by producers, mutators, and observers;

C. and no representation exposure occurs,

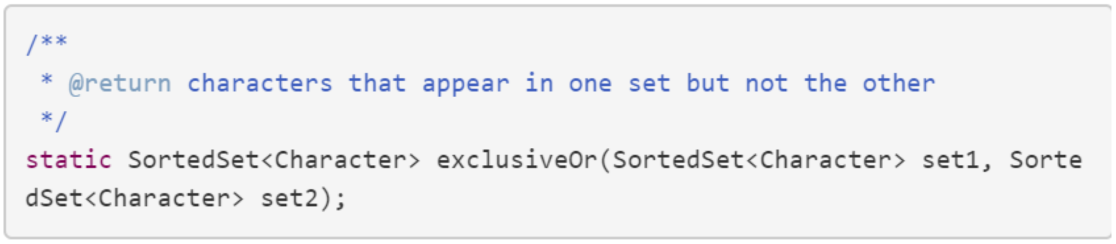
拓展：ADT invariants replace preconditions

An enormous advantage of a well-designed abstract data type is that it encapsulates and enforces properties that we would otherwise have to stipulate in a precondition.（良好设计的 ADT 的一个大优点在于我们可以使用它将本该写在前置条件中的限制封装起来）

也就是使用特定类型的 ADT 来代替规约中对于参数的限制，不依赖于客户对规约 precondition 的保证，而是在参数类型上直接进行限制，提示错误。



```
/**
 * @param set1 is a sorted set of characters with no repeats
 * @param set2 is likewise
 * @return characters that appear in one set but not the other,
 *  in sorted order with no repeats
 */
static String exclusiveOr(String set1, String set2);
```



```
/**
 * @return characters that appear in one set but not the other
 */
static SortedSet<Character> exclusiveOr(SortedSet<Character> set1, Sorte
dSet<Character> set2);
```

在此例中就是使用 SortedSet 来进行限制，以代替规约的前置条件。

This hits all our targets:

it’s safer from bugs, because the required condition (sorted with no repeats) can be enforced in exactly one place, the SortedSet type, and because Java static checking comes into play, preventing values that don’t satisfy this condition from being used at all, with an error at compile-time.

it’s easier to understand, because it’s much simpler, and the name SortedSet conveys what the programmer needs to know.

it’s more ready for change, because the representation of SortedSet can now be changed without changing exclusiveOr or any of its clients.

3.3.8 Format of an ADT（ADT 的格式）

Documenting the AF, RI, and safety from rep exposure

document the abstraction function，rep invariant and a rep exposure safety argument in the class, using comments right where the private fields of the rep are declared.

以注释的形式，将抽象函数（AF），表示不变量（RI），表示暴露的安全措施（a rep exposure safety argument）写在 ADT 的表示量之后。

Ex：

```java
// Immutable type representing a tweet.
public class Tweet {

    private final String author;
    private final String text;
    private final Date timestamp;

    // Rep invariant:
    //   author is a Twitter username (a nonempty string of letters, dig
its, underscores)
    //   text.length <= 280
    // Abstraction function:
    //   AF(author, text, timestamp) = a tweet posted by author, with co
ntent text,
    //                                 at time timestamp
    // Safety from rep exposure:
    //   All fields are private;
    //   author and text are Strings, so are guaranteed immutable;
    //   timestamp is a mutable Date, so Tweet() constructor and getTime
stamp()
    //        make defensive copies to avoid sharing the rep's Date obje
ct with clients.

    // Operations (specs and method bodies omitted to save space)
    public Tweet(String author, String text, Date timestamp) { ... }
    public String getAuthor() { ... }
    public String getText() { ... }
    public Date getTimestamp() { ... }
}
```

```java
// Immutable type representing a rational number.
public class RatNum {
    private final int numerator;
    private final int denominator;

    // Rep invariant:
    //   denominator > 0
    //   numerator/denominator is in reduced form, i.e. gcd(|numerator|,
denominator) = 1
    // Abstraction function:
    //   AF(numerator, denominator) = numerator/denominator
    // Safety from rep exposure:
    //   All fields are private, and all types in the rep are immutable.

    // Operations (specs and method bodies omitted to save space)
    public RatNum(int n) { ... }
    public RatNum(int n, int d) { ... }
    ...
}
```

Write an ADT spec

编写一个规约，只可以关注使用者可见的部分，这包括参数，返回值，可能抛出的异常等等，它应该是抽象域的值而非表示域。

注：规约使用/** */，注释使用/**/或//。一定要区分开，因为 java 会自动提取除/** */。