Recap about the Utility and Non-Utility Classes

utility classes:
- cannot be instantiated
- all methods and/or fields are static
- e.g., the class Toolbox

non-utility classes:
- can be instantiated
- may include both non-static and static methods and/or fields
- e.g., the class Integer. Integer.MAX_INT is a static field and Integer.parseInt is a static method. The class has no non-static fields and toString() is a non-static method

Recap about the Client View

- implementers: offers services in the form of classes (utility and non-utility classes)
- clients: make use of the services offered by implementers, subject to the Pre and Post conditions
Recap about the PRE and POST Conditions

- **PRE** – condition(s) that the client must satisfy
- **POST** – condition(s) that the implementer must satisfy

- PRE and POST conditions are used to establish correctness.
  - Do the services function according to their specification
  - This is done during development, before services are released and go into production

- PRE and POST conditions eliminate redundancy.
  - Is the CLIENT or IMPLEMENTER responsible for checking the value of the parameters being passed to methods?
  - In the absence of information, both might do this. This could cause an app to be inefficient.

PRE and POST: the Client View

- the Pre and Post conditions are described in the API
- the Pre and Post conditions are sometimes formulated in terms of a boolean expression that must evaluate to true
PRE Example

suppose we have the method
public String fraxas(int num)

- PRE written version:
  num must be strictly greater than 0
- for the PRE to be met, the written description must hold

- boolean expression version:
  num > 0
- for the PRE to be met, num must be such that
  num > 0 == true

---

PRE Example

suppose we have the method
public String saxa(int num)

- PRE written version:
  num can be any int
- for the PRE to be met, the written description must hold

- boolean expression version:
  true
- for the PRE to be met, num must be s.t.
  true == true (or just true)

...but any value of num will satisfy this, since the boolean expression does not even depend on num

...but it will always be trivially true, since the compiler will do the type checking

see sec 2.3.3 for further review
PRE and POST: the Client View

- in Java, most often the PRE is true
  - this means the client needs simply to provide the parameter values (and nothing further in terms of conditions on those values)

- in Java, it quite often happens that the POST consists of both:
  1. a specification of the return, and
  2. a specification of the condition under which exceptions are thrown

Example

```
substring
public String substring(int beginIndex)
Returns a new String that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.
Examples:
"unhappy".substring(2) returns "happy"
"Warholic".substring(3) returns "holic"
"emptiness".substring(7) returns "" (an empty String)
```

- no precondition is specified
- PRE is true

- "returns" and "throws" are parts of the post condition