A Representation Invariant is a property of an ADT that must be true at all times for the ADT’s state to be considered valid. Formally, a representation invariant is a boolean expression - something that evaluates to “true” or “false”.
Representation Invariant: The Heap Property

For example, in a max-heap: for every node $N$, $N$’s child nodes must have keys that are smaller than $N$. (the largest key is in the root node.)
Typically, you can use inductive reasoning to prove that your Representation Invariant will hold. For heap order property:

- Base Case:
- Inductive Hypothesis:
- Inductive Step:
A Representation Invariant is an assumption that you make about the code that you’re writing - but sometimes, your assumptions can be wrong. Representation Invariants should give you a good idea of what you should be asserting in your code.
In the code for your heap, you can write a function `bool checkHeapProperty()` to verify if a heap is valid.

class Heap {
    Heap(); // check here
    void insert(item); // check here (before and after)
    void deleteMax(); // check here (before and after)
    item getMax();
};
An Abstraction Function is a mapping of the *code implementation* that implements an ADT to the *abstract model* of that same ADT that we have in our heads.
Heap Abstraction Function

Representing a heap with \( k \) elements using an array of size \( k \):

\[
AF(r) = \text{a heap such that for all nodes } n \text{ in heap,}
\]

\[
8, 7, 5, 4, 3, 4, 1, 2, 3, 2
\]
Bubble Tea is a tea-based drink originating from Taiwan. It consists of:

- a base (green or black tea),
- milk (or no milk),
- a flavour (e.g. strawberry, blueberry, taro, kiwi, chocolate, coconut...)
- extra toppings (e.g. tapioca balls, various flavours of jelly)
In code, we can represent a Bubble Tea object like so:

class BubbleTea {
    int base_;  // 0 = green, 1 = black
    bool hasMilk_;
    string flavour_;
    string[] extraToppings_;  
};
Representation Invariant:
What do you expect must always be true of a serving of Bubble Tea for it to be valid (i.e. for them to be able to make it and sell it to you?)
Abstraction Function: How does the code version of a Bubble Tea ADT map to an actual serving of Bubble Tea?
Representation Invariant:

base_ must be 0 or 1,
flavour_ must come from a list of valid flavours (the menu)
extraToppings_ must also come from the menu, and
extraToppings_.size <= MAX_TOPPINGS

Abstraction Function:

AF(r) =
Base = B(base_) where B(0) = green, and B(1) = black.
Has Milk if hasMilk_ is true,
Flavour = flavour_, and,
Toppings = all elements in extraToppings_.[].
Consider some representation of a Company:

class Company{
    class Employee{
        string name;
        int num_subord;
        Employee* subord[];
    };
    Employee owner;
};

What would the abstraction function be for mapping this Company ADT to a stafflist of some company. (hint: think recursively)
Abstraction Function:

\[ AF(r) = \{ r\text{.name} \} \cup \{ \} \cup AF(*r\text{.subord}[0]) \cup ... \cup AF(*r\to subord[num\_subord-1]) \]

A full stafflist of the company corresponds to \( AF(\text{owner}) \).
The End