* **Container:** An object holding a collection of elements, e.g. *Vector*.
* **Abstract Type:** A type that completely insulates a user from implantation details.
* The interface is decoupled from the implementation.
* An abstract container class does not have a constructor.
* An abstract container class does have a destructor and that destructor is virtual.
* **rvalue:** This word is intended to complement lvalue, which means “something that can appear on the left-hand side of the assignment.
* *rvalue* is a value that one can’t assign to, such as an integer returned by a function call.
* *rvalue reference* is a reference to something that nobody else can assign to.
* **Advice –**
* Define classes to represent application concepts directly in code.
* Use concrete classes to represent simple concepts and performance-critical components.
* Use resource handles and RAII (resource acquisition is initialisation) to manage resources.
* Use abstract classes as interfaces when complete separation of interface and implementation is needed.
* Use class hierarchies to represent concepts with inherent hierarchical structure.
* When designing a class hierarchy, distinguish between implementation inheritance and interface inheritance.
* Control construction, copy, move, and destruction of objects.
* Return containers by value (relying on move for efficiency).
* Provide strong resource safety, i.e. never leak anything that you think of as a resource.
* Use containers, defined as resource handle templates, to hold collections of values of the same type.
* Use function templates to represent general algorithms.
* Use function objects, including lambdas, to represent policies and actions.
* Use type and template aliases to provide a uniform notation for types that may vary among similar types or among implementations.