|  |
| --- |
| **Differences between Abstract Classes and Interfaces.** |
|  |
|  |

Abstract classes and interfaces both are primarily used for abstraction, however they are very different from each other. Both are a core part of Java but have their own application uses. While both terms may be synonymous with each other, they cannot be used interchangeably.

Abstract classes, unlike interfaces, are classes. They are more expensive to use, because there is a look-up to do when you inherit from them. Abstract classes look a lot like interfaces, but they have something more: You can define a behavior for them. **Interfaces can be extended only by interfaces. Classes have to implement them instead of extend.** An Abstract class doesn't provide full abstraction but an interface does provide full abstraction. Using Abstract, we can’t achieve multiple inheritance but using an Interface we can achieve multiple inheritance. **An interface is an empty shell** as well as a contract. The interface can't do anything, as it's just a pattern.

Interface or a protocol is a common way that allows unrelated objects to communicate with each other. It represents an agreed-upon behavior that facilitates the interaction between the two objects. An abstract class is a class of objects that cannot be instantiated or set up an instance of an object. This class may have no implementation or can have incomplete implementation.

|  |
| --- |
|  |

**Why Unit Testing is important and what benefits it adds.**

Unit testing is a software testing method by which individual units of source code are tested to determine whether they are fit for use. A unit is the smallest possible testable software component. Usually, it performs a single cohesive function. A unit is small, so it is easier to design, execute, record, and analyze test results for than larger chunks of code are. Defects revealed by a unit test are easy to locate and relatively easy to repair.

The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly. It isolates the smallest piece of testable software from the remainder of the code and determines whether it behaves exactly as you expect. Unit testing has proven its value in that a large percentage of defects are identified during its use. It allows automation of the testing process, reduces difficulties of discovering errors contained in more complex pieces of the application, and enhances test coverage because attention is given to each unit.

Unit testing provides numerous benefits including finding software bugs early, facilitating change, simplifying integration, providing a source of documentation, and many others.

**Benefits of Unit Testing**

## **1. Makes the Process Agile**

One of the main benefits of unit testing is that it makes the coding process more Agile. When you add more and more features to a software, you sometimes need to change old design and code. However, changing already-tested code is both risky and costly Unit testing really goes hand-in-hand with agile programming of all flavors because it builds in tests that allow you to make changes more easily. In other words, unit tests facilitate safe refactoring.

## **2. Quality of Code**

Unit testing improves the quality of the code. It identifies every defect that may have come up before code is sent further for integration testing. Writing tests before actual coding makes you think harder about the problem. It exposes the edge cases and makes you write better code.

## **3. Finds Software Bugs Early**

Issues are found at an early stage. Since unit testing is carried out by developers who test individual code before integration, issues can be found very early and can be resolved then and there without impacting the other pieces of the code. This includes both bugs in the programmer’s implementation and flaws or missing parts of the specification for the unit.

## **4. Facilitates Changes and Simplifies Integration**

Unit testing allows the programmer to refactor code or upgrade system libraries at a later date and make sure the module still works correctly. Unit tests detect changes that may break a design contract. They help with maintaining and changing the code. Unit testing reduces defects in the newly developed features or reduces bugs when changing the existing functionality. Unit testing verifies the accuracy of each unit. Afterward, the units are integrated into an application by testing parts of the application via unit testing. Later testing of the application during the integration process is easier due to the verification of the individual units.

## **5. Provides Documentation**

Unit testing provides documentation of the system. Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit’s interface (API).

## **6. Debugging Process**

Unit testing helps simplify the debugging process. If a test fails, then only the latest changes made in the code need to be debugged.

## **7. Design**

Writing the test first forces you to think through your design and what it must accomplish before you write the code. This not only keeps you focused; it makes you create better designs. Testing a piece of code forces you to define what that code is responsible for. If you can do this easily, that means the code’s responsibility is well-defined and therefore that it has high cohesion.

## **8. Reduce Costs**

Since the bugs are found early, unit testing helps reduce the cost of bug fixes. Bugs detected earlier are easier to fix because bugs detected later are usually the result of many changes, and you don’t really know which one caused the bug.

**References**:

<https://beginnersbook.com/2013/05/abstract-class-vs-interface-in-java/>

<http://www.differencebetween.info/difference-between-interface-and-abstract-class-in-java>

<https://dzone.com/articles/top-8-benefits-of-unit-testing>

<http://www.differencebetween.net/technology/difference-between-abstract-class-and-interface-in-java/>