1. **Predefined functions in collection.**

|  |  |
| --- | --- |
| [add(Object)](https://www.geeksforgeeks.org/collection-add-method-in-java-with-examples/) | This method is used to add an object to the collection. |
| [addAll(Collection c)](https://www.geeksforgeeks.org/collections-addall-method-in-java-with-examples/) | This method adds all the elements in the given collection to this collection. |
| [clear()](https://www.geeksforgeeks.org/collection-clear-method-in-java-with-examples/) | This method removes all of the elements from this collection. |
| [contains(Object o)](https://www.geeksforgeeks.org/collection-contains-method-in-java-with-examples/) | This method returns true if the collection contains the specified element. |
| containsAll(Collection c) | This method returns true if the collection contains all of the elements in the given collection. |
| equals(Object o) | This method compares the specified object with this collection for equality. |
| hashCode() | This method is used to return the hash code value for this collection. |
| [isEmpty()](https://www.geeksforgeeks.org/collection-isempty-method-in-java-with-examples/) | This method returns true if this collection contains no elements. |
| iterator() | This method returns an iterator over the elements in this collection. |
| [max()](https://www.geeksforgeeks.org/collections-max-method-in-java-with-examples/) | This method is used to return the maximum value present in the collection. |
| parallelStream() | This method returns a parallel Stream with this collection as its source. |
| remove(Object o) | This method is used to remove the given object from the collection. If there are duplicate values, then this method removes the first occurrence of the object. |
| removeAll(Collection c) | This method is used to remove all the objects mentioned in the given collection from the collection. |
| removeIf(Predicate filter) | This method is used to remove all the elements of this collection that satisfy the given [predicate](https://www.geeksforgeeks.org/mathematic-logic-predicates-quantifiers/). |
| retainAll(Collection c) | This method is used to retain only the elements in this collection that are contained in the specified collection. |
| size() | This method is used to return the number of elements in the collection. |
| spliterator() | This method is used to create a [Spliterator](https://www.geeksforgeeks.org/java-program-to-convert-iterator-to-spliterator/) over the elements in this collection. |
| stream() | This method is used to return a sequential Stream with this collection as its source. |
| toArray() | This method is used to return an array containing all of the elements in this collection. |

1. **underlying ds for linked list,arraylist.**

\*)ArrayList is a resizable array implementation in java.

The backing data structure of ArrayList is an array of Object class.

When creating an ArrayList you can provide initial capacity then the array is

declared with the given capacity. The default capacity value is 10.

\*)Linked List is a linear data structure. Unlike arrays, linked list elements are

not stored at the contiguous location, the elements are linked using pointers as shown below.

In Java, LinkedList can be represented as a class and a Node as a separate class

1. **Advantages and disadvantages of array list**

*Advantages:*

1. ArrayList is variable length.
2. Add any type of data into ArrayList.
3. Traverse in both directions.
4. Insert and remove elements also at particular position of ArrayList.
5. ArrayList allows multiple null values.
6. ArrayList allows to add duplicate elements.
7. ArrayList has many methods to manipulate stored objects.
8. When ArrayList exceeds its capacity, then its size increases by 50%.
9. Retrieval is faster in ArrayList

*Disadvantages:*

1. If a data entry is added to or removed from an array-based list, data needs to be shifted to update the list.
2. **What are iterators and cursors.**

If we want to get objects one by one from the [Collection](https://codezup.com/why-do-we-need-collections-framework-in-java/), then we should go for **Cursors**. Iterator is a type of cursor

1. **what are list iterator,iterator,enumerator with code.**

Enumeration Cursor

* We can use Enumeration to get Objects one by one from the legacy collection objects.
* We can create an Enumeration object by using elements () method.

public Enumeration elements ();

Enumeration e =v.elements();

Using vector object.

Enumeration interface defines the following two methods:

1. Public boolean hasMoreElements( );
2. Public Object nextElement();

Limitation of Enumeration:

1. We can apply the Enumeration concept only for legacy classes and it is not a universal cursor.
2. By using Enumeration, we can get only read access and we cannot perform remove operation.
3. To overcome these limitations there is a new concept introduced which is known as the Iterator in Java.

Iterator Cursor

* We can use the Iterator to get objects one by one from any collection object.
* We can apply the Iterators concept for any collection object and due to this reason, it is known as Universal Cursor.

1. While iterating the objects by Iterator in Java we can perform both read and remove operations.

How we can get Iterator Object in Java?

We can get the Iterator object by using the Iterator() method of Collection interface.

Public Iterator iterator();

Iterator itr=c.iterator();

Iterator interface defines the following three methods.

1. public boolean hasNext();
2. public object next();
3. public void remove();

**Limitation of Java Iterator**:

1. Both Enumeration and Iterator are single direction cursor only. That is, we can always move only forward direction and we can not move to the backward direction.
2. While iterating by Iterator we can perform **only read and remove operations**and we cannot perform replacement and addition of new Objects.
3. To overcome these limitations a new concept is introduced which is known as **ListIterator.**

### 3. [ListIterator](https://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html" \t "_blank):

1. **ListIterator** is the child interface of the **Iterator**.
2. By using ListIterator we can move either to the **forward direction or to the backward direction** that is it is a **bi-directional cursor**.
3. While iterating by ListIterator we can perform **replacement** and **addition** of new objects in addition to **read** and **remove** operations.

By using listIterator method we can create listIterator object.

Public ListIterator listiterator();

ListIterator itr=c.Listiterator();

#### Methods in ListIterator

ListIterator interface defines the following 9 methods:

1. **public boolean hasNext();**
2. **public Object next(); //forward**
3. **public int nextIndex();**
4. **public boolean hasPrevious();**
5. **public Object previous(); // backward**
6. **public int previousIndex();**
7. **public void remove ();**
8. **public void set (Object new);**
9. **public void add(Object new);**

Graphical user interface, application

Description automatically generated

1. **List down the security breaches than can happen in front end backend.**

CSRF

[Cross-site request forgery](https://owasp.org/www-community/attacks/csrf) or CSRF is a browser exploit where an attacker tricks someone into executing unwanted actions against a website to which their browser is already authenticated.

This could be your social media, bank, or any other account that somehow saves your credentials so that you don’t constantly have to log in on every session.

The attacker uses social engineering to get the user to click on a malicious link or to open an email or webpage with an embedded transparent 1px image that contains the link.

The unknowing browser loads the page sends the request and executes the malicious action. The underlying issue comes from the attacker guessing the URL structure, which could be fixed by associating a random hash with each session and credential cookies, ensuring that cookies won’t be used outside that session.

CSS injection

Unlike the previous vulnerability, this exploit doesn’t require a user. It simply requires developers to use a CSS file that has a secret malicious import statement.

You might say: “Who would be stupid enough to do something like that?”. But remember that in the age of npm, where we use a few packages that have dozens of sub dependencies, such a mistake can happen easily.

CSS imports can execute javascript code. For example, check out this proof-of-concept [CSS keylogger](https://github.com/maxchehab/CSS-Keylogging) that can leak passwords and other forms to the attacker.

A lot of this can be prevented by self-hosting such files on your servers, but it might require using an actual [vulnerability management tool](https://debricked.com/tools/security/) to ensure your developers stay on top of any vulnerabilities in such files.

Remote imports from untrusted parties

This one is similar to the previous one, but it has to do with any package or file import. Finding a library that does just what you want is great, but often we don’t check every little detail of said library. That can, later on, lead to us unknowingly introducing a vulnerability into our system.

Such things are bad when they happen to backends, but they are worse in the frontend.

For communication between components you own, you can simply use a [service mesh](https://www.buzzwrd.me/index.php/2021/02/17/service-mesh-works-great-if-you-can-accept-its-trade-offs/) or similar solution to fine-tune what each of the components can access on the network. Especially when such imports are processed client-side.

This means that the user’s browser simply downloads and executes the code on that import. If the remote URL code changes, it will change for all users without us noticing or doing anything about it.

While this scenario might sound unlikely to you, there are both malicious developers out there and people willing to take advantage of a legitimate developer’s work. [We’ve previously written](https://debricked.com/blog/how-to-evaluate-npm-vulnerabilities/) about the risk of popular package ownership being taken over by a malicious entity.

As most programming these days depends on freely shared open-source code in the form of snippets or modules, such attacks will just become more and more dangerous.

Issues with authentication, cookies, and forms

Badly written forms, as well as any issue with authentication and authorization mechanisms, can be very dangerous if left in the open.

Issues with such code can be rather dangerous but are a rather large topic. Instead of going over them in detail, we will simply point you to a few comprehensive articles on the topic:

* [How to spot issues with authentication and authorization](https://auth0.com/blog/what-is-broken-authentication/)
* [Preventing broken authentication](https://openclassrooms.com/en/courses/5162996-secure-your-web-application-with-owasp/6122341-prevent-broken-authentication)
* [How to design secure forms](http://freecodecamp.org/news/security-for-the-front-end-developer/)

### XSS

This is a rather widespread set of vulnerabilities that allow an attacker to pass malicious code as input to some input field in the app.

If the app doesn’t sanitize this malicious input and just enters/displays it as is, it can lead to exposing data or performing unwanted actions.

Check out this[dummy XSS vulnerable application](https://medium.com/@tattwei46/what-is-sql-injection-and-xss-2a3f2e7ea0d), by exploiting similar vulnerabilities you could leak data, modify it, create code snippets that execute when other users try to use the app, and many more malicious actions.

The major problem is managing such vulnerabilities when you depend on many modules and open-source code, as different versions can hold different vulnerabilities, and depending on an old vulnerable version can end up compromising your application.

The issue here is that it’s easy to google the most recent information and most notable information regarding some package or module; it may be the case that vulnerabilities introduced in newer versions don’t affect your older dependency.

There are even some cases where due to legacy reasons you have to run old versions of the software.

This means that you can end up with the worst of both worlds. Where on the one hand you may depend on software with vulnerabilities that need to be updated or fixed, and on the other hand you might be under pressure to needlessly upgrade some component that is currently under bad press due to a major vulnerability that affects only versions of the component newer than the one you use.

### Using components with vulnerabilities

Another way you can seriously harm the security of your application is by using components that have known security issues.

For example, let’s look at [react-oauth-flow](https://www.npmjs.com/package/react-oauth-flow), an npm module focusing on providing oAuth functionality to react apps. It has a steady 1000 downloads per week and quite some stars on GitHub.

So, it must be a good easy-to-use option for adding oAuth to a react app, right? Well, it would be, except that it was found out that it leaked secrets in the frontend, allowing attackers to compromise server credentials.

This is just an example, and similar vulnerabilities are likely hidden in many popular modules out there. Fortunately, this was an open-source module which meant that there were many eyes checking for bugs and reporting them, meaning that the problem was relatively quickly mitigated.

But unless you have an explicit process for reviewing your components and sources for new vulnerabilities, there isn’t much you can do but wait to get hacked.

As most people generally don’t derive pleasure from checking for 3rd party and open-source component vulnerabilities, this task is best automated or delegated to a security expert.

**7. Owasp**

1. Broken Access Control

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

### 2. Cryptographic Failures

Many web applications and APIs do not properly protect sensitive data with strong encryption. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data must be encryption at rest and in transit, using a modern (and correctly configured) encryption algorithm.

### 3. Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

### 4. Insecure Design

Pre-coding activities are critical for the design of secure software. The design phase of you development lifecycle should gather security requirements and model threats, and development time should be budgeted to allow for these requirements to be met. As software changes, your team should test assumptions and conditions for expected and failure flows, ensuring they are still accurate and desirable. Failure to do so will let slip critical information to attackers, and fail to anticipate novel attack vectors.

### 5. Security Misconfiguration

Your software is only as secure as you configure it to be. Using ad hoc configuration standards can lead to default accounts being left in place, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

### 6. Vulnerable and Outdated Components

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

### 7. Identification and Authentication Failures

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.

### 8. Software and Data Integrity Failures

Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure deployment pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations.

### 9. Security Logging and Monitoring Failures

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

### 10. Server-Side Request Forgery

Server-Side Request Forgery (SSRF) flaws occur whenever a web application fetches a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

1. **java program for divide by zero.**

Graphical user interface, text, application

Description automatically generated

1. [**Difference between ArrayIndexOutOfBoundsException and IndexOutOfBoundsException?**](https://stackoverflow.com/questions/34266174/difference-between-arrayindexoutofboundsexception-and-indexoutofboundsexception)

[IndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/IndexOutOfBoundsException.html) :Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException are two classes, which have implemented IndexOutOfBoundsException.

[ArrayIndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/ArrayIndexOutOfBoundsException.html): Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

1. **can catch be return without try**

No, it is not possible because the catch block would be useless as there is no piece of code to throw an exception.

1. **can try be return catch**

Yes, we can have try without catch block by using finally block. You can use try with finally. As you know finally block always executes even if you have exception or return statement in try block except in case of System.

1. **can finally be return without try catch**

Wemust have a try block with a finally block. The try block defines which lines of code will be followed by the finally code. If an exception is thrown prior to the try block, the finally code will not execute.

1. **difference b/w comparable and comparator**

Comparable

* Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.
* Comparable **affects the original class**, i.e., the actual class is modified.
* Comparable provides **compareTo() method** to sort elements.
* Comparable is present in **java.lang** package.
* We can sort the list elements of Comparable type by **Collections.sort(List)** method.

Comparator

* The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
* Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
* Comparator provides **compare() method** to sort elements.
* A Comparator is present in the **java.util** package.
* We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.