**1) Type-safety:** We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

Without Generics, we can store any type of objects.

1. List list = **new** ArrayList();
2. list.add(10);
3. list.add("10");
4. With Generics, it is required to specify the type of object we need to store.
5. List<Integer> list = **new** ArrayList<Integer>();
6. list.add(10);
7. list.add("10");// compile-time error

**2) Type casting is not required:** There is no need to typecast the object.

Before Generics, we need to type cast.

1. List list = **new** ArrayList();
2. list.add("hello");
3. String s = (String) list.get(0);//typecasting
4. After Generics, we don't need to typecast the object.
5. List<String> list = **new** ArrayList<String>();
6. list.add("hello");
7. String s = list.get(0);

**3) Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

1. List<String> list = **new** ArrayList<String>();
2. list.add("hello");
3. list.add(32);//Compile Time Error

Wildcard in Java Generics

The ? (question mark) symbol represents the wildcard element. It means any type.

### Upper Bounded Wildcards

### List<? extends Number>

suitable for a list of type Number or any of its subclasses

## Lower Bounded Wildcards

List<? **super** Integer>

 we want to write the method for the list of Integer and its supertype (like Number, Object).