Java Generics Cheatsheet

Java generics provide a way to create classes, interfaces, and methods that operate on types as parameters, allowing for type safety and reusability. This cheatsheet explains Java generics, wildcards, and bounded types with examples of generic classes and methods.

Table of Contents

Introduction to Generics

Generics enable classes, interfaces, and methods to operate on **type parameters**. They provide type safety and allow code to be more reusable. Generics were introduced in Java 5.

```
// Without generics (pre-Java 5)
List list = new ArrayList();
list.add("Hello");
String str = (String) list.get(0); // Type casting needed

// With generics
List<String> strList = new ArrayList<>();
strList.add("Hello");
String str = strList.get(0); // No type casting needed
```

Generic Classes

Generic classes are classes that use type parameters.

```
class Box<T> {
    private T value;

    public Box(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}

Box<Integer> intBox = new Box<>(42);
int value = intBox.getValue(); // No casting needed
```

Generic Methods

Generic methods are methods that have their own type parameters.

```
public <T> T getFirst(List<T> list) {
    if (list.isEmpty()) {
        throw new IllegalArgumentException("List is empty");
    }
    return list.get(0);
}

List<String> names = List.of("Alice", "Bob", "Charlie");
String first = getFirst(names);
```

Wildcards

Wildcards allow you to write methods and classes that operate on unknown types.

- `<?>` (Unbounded wildcard): Accepts any type.
- `<? extends T>` (Upper bounded wildcard): Accepts types that are subtypes of `T`.
- `<? super T>` (Lower bounded wildcard): Accepts types that are superclasses of `T`.

```
public void printList(List<?> list) {
    for (Object item : list) {
        System.out.println(item);
    }
}

List<String> strings = List.of("A", "B", "C");
List<Integer> integers = List.of(1, 2, 3);

printList(strings);
printList(integers);

List<? extends Number> numbers = List.of(1, 2.5, 3L);
```

Bounded Types

Bounded types restrict type parameters to specific hierarchies.

```
class NumberBox<T extends Number> {
    private T value;

    public NumberBox(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}

NumberBox<Integer> intBox = new NumberBox<>(42);
NumberBox<Double> doubleBox = new NumberBox<>(3.14);

// NumberBox<String> strBox = new NumberBox<>("Hello"); // Compile error
```

In this cheatsheet, we've covered the basics of Java generics, including generic classes, methods, wildcards, and bounded types. Generics are a fundamental feature that enhances type safety and code reusability in Java.