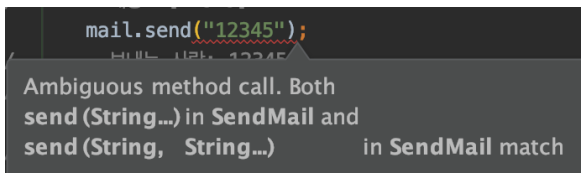
아이템 53 가변인수는 신중히 사용하라

▼ **Vararges**

- 자바 5부터 가변인수(Vararags)를 사용할 수 있다.
- 가변인수라는 것은 필요에 따라 매개변수를 가변적으로 조정할 수 있는 기술이다.
- 가변인수가 없었을 때는 컬렉션이나 배열을 이용해 가변인수를 대체했었다.
- 컴파일러에서 가변인수를 배열로 변환해준다.
- 가변인수 메서드를 오버로딩한 여러 메서드가 있다면, 컴파일러는 메소드를 구분하지 못해 컴파일 에러가 발생한다. → 가변인수 메소드를 오버로딩 하지 않는 것이 좋다.




```
class SendMail {
    private String sender;
    private List<Object> content = new ArrayList<>();

    void print(){
        System.out.println("보내는 사람: "+sender);
        System.out.println("내용: "+content);
    }
 void send(String... args) {
        sender = "ANONYMOUS";
        for (Object s : args) {
            content.add(s);
        }
        print();
    }

    void send(String email, String... args) {
        sender = email;
        for (Object s : args) {
            content.add(s);
        }
        print();
    }

    public static void main(String[] args) {
        SendMail mail = new SendMail();
        mail.send(); // OK
        mail.send("12345");
        mail.send("aaaa@email.com", "HI");
    }
 }
```

- 가변인수 메서드는 명시한 타입의 인수를 0개 이상 받을 수 있다.
- 가변 인수 메서드를 호출하면 가장 먼저 인수의 개수와 길이가 같은 **배열**을 만들고 인수들을 이 배열에 저장하여 가변 인수 메서드에 건네준다.

```
static int sum(int... args) {
        int sum = 0;
        for (int arg : args)
            sum += arg;
        return sum;
    }
// 간단한 가변인수 활용 예
```

인수가 1개 이상이어야 할 때

- 최솟값을 찾는 메서드 인데 인수를 0개만 받을 수도 있도록 설계하는 건 좋지 않다.
- 인수 개수는 런타임에 (자동 생성된) **배열**의 길이로 알 수 있다.

```
static int min(int... args) {
        if (args.length == 0)
            throw new IllegalArgumentException("Too few arguments");
        int min = args[0];
        for (int i = 1; i < args.length; i++)
            if (args[i] < min)
                min = args[i];
        return min;
    }
```

- 문제점 : 인수를 0개만 넣어 호출하면 런타임에 실패한다, 코드도 지저분하다
- args 유효성 검사를 명시적으로 해야하고, min의 초깃값을 Integer.MAX_VALUE로 설정하지 않고는 for-each 문도 사용할 수 없다.

매개변수를 2개 받도록하면 앞의 문제가 사라진다.

```
static int min(int firstArg, int... remainingArgs) {
        int min = firstArg;
        for (int arg : remainingArgs)
            if (arg < min)
                min = arg;
        return min;
    }
```

- 가변인수의 등장으로 인해 printf가 등장했고, 핵심 리플렉션기능(아이템 65)도 재정비되었다.

```
 * @since  1.5
 */
public PrintStream printf( @NotNull String format, Object ... args) {
    return format(format, args);
}
```

가변인수의 대표적인 예 PrintStream의 printf()
ex : System.out.printf("%s, %d년 %d월 %d일", "today", 2021, 6, 27);

```
 */
@CallerSensitive
@ForceInline // to ensure Reflection.getCallerClass optimization
@HotSpotIntrinsicCandidate
public Object invoke(Object obj, Object... args) Complexity is 6 It's time to do s
    throws IllegalAccessException, IllegalArgumentException,
       InvocationTargetException
{
    if (!override) {
        Class<?> caller = Reflection.getCallerClass();
        checkAccess(caller, clazz,
                    Modifier.isStatic(modifiers) ? null : obj.getClass(),
                    modifiers);
    }
    MethodAccessor ma = methodAccessor;             // read volatile
    if (ma == null) {
        ma = acquireMethodAccessor();
    }
    return ma.invoke(obj, args);
}
```

가변인수 대표적인 예 reflection invoke()

성능에 민감한 상황이라면 가변인수가 걸림돌이 될 수 있다.

- 가변인수 메서드는 호출될 때마다 **배열**을 새로 하나 할당하고 초기화 한다.

가변인수의 유연성이 필요할 때 사용할 수 있는 패턴

1. 메서드 호출의 95%가 인수를 N개 이하로 사용할 때
2. 인수가 0개인것부터 N+1개인 것까지 총 N+2개를 다중정의(overloading) 한다.

   마지막 다중정의 메서드가 인수 N+1개 이상인 5%의 호출을 담당하게 한다.

```
public void foo(){}
public void foo(int a1){}
public void foo(int a1, int a2){}
public void foo(int a1, int a2, int a3){} // 여기까지 95 % 담당
public void foo(int a1, int a2, int a3, int ...rest){} // 나머지 5% 담당
```

- **EnumSet**의 정적 팩터리도 이 기법을 사용해 열거 타입 집합 생성 비용을 최소화한다.

static <E extends Enum<E>> EnumSet<E>	of(E e) Creates an enum set initially containing the specified element.
static <E extends Enum<E>> EnumSet<E>	of(E first, E... rest) Creates an enum set initially containing the specified elements.
static <E extends Enum<E>> EnumSet<E>	of(E e1, E e2) Creates an enum set initially containing the specified elements.
static <E extends Enum<E>> EnumSet<E>	of(E e1, E e2, E e3) Creates an enum set initially containing the specified elements.
static <E extends Enum<E>> EnumSet<E>	of(E e1, E e2, E e3, E e4) Creates an enum set initially containing the specified elements.
static <E extends Enum<E>> EnumSet<E>	of(E e1, E e2, E e3, E e4, E e5) Creates an enum set initially containing the specified elements.

성능의 이유로 5개 이상의 요소를 추가해야 하는 경우 가변인수사용(E... rest)

  - EnumSet은 비트필드(아이템 36)을 대체하면서 성능까지 유지해야 하므로 적절하게 활용한 예시다.

핵심 정리

- 인수 개수가 일정하지 않은 메서드를 정의해야 한다면 가변인수가 반드시 필요하다.
- 메서드를 정의할 때 필수 매개변수는 가변인수 앞에 두고, 가변인수를 사용할 때는 성능 문제까지 고려하자.