

Estructuras de Datos

Genéricos

Adriana Collaguazo, Mg.

Agenda

1. Tipos de Datos Genéricos
 1. Por qué
 2. Qué
 3. Cómo (en Java)

Introducción

Motivación

Clase `Operation`

4 métodos estáticos para dividir un número `n`:

```
public static int divideByTwo (int n)
```

```
public static int divideByThree (int n)
```

```
public static int divideByFour (int n)
```

```
public static int divideByFive (int n)
```

¿Cuál es el problema con esta estrategia?

Problemas

Difícil de mantener

No es escalable

Es muy específica

Cuál es la solución

Solución

Escribir un solo método

Utilizar un parámetro para el divisor de la operación

```
public static int divideBy (int n, int divisor)
```

Qué ganamos con esto?

Ahora podemos enviar un argumento que defina el divisor

Logramos una solución **más genérica** (menos específica)

Y escalable

Parametrización de Datos

Permite modificar el comportamiento de un código, sin necesidad de reescribirlo

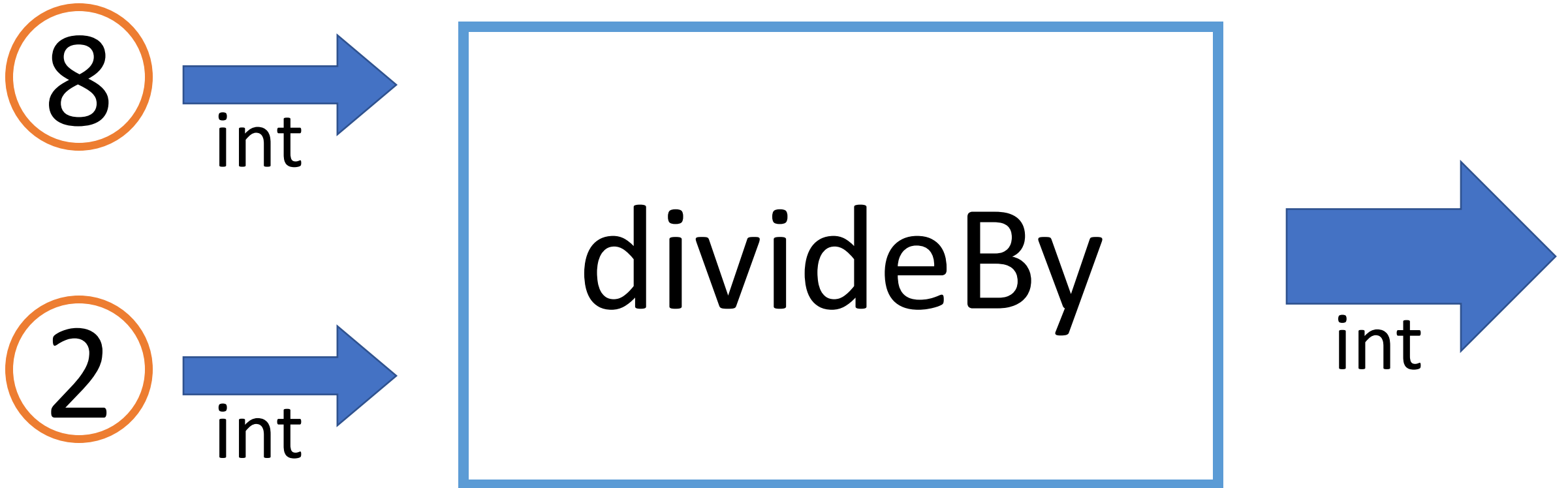
Es decir, permite la programación de rutinas genéricas

Permite cambiar qué piezas de información enviamos a un método

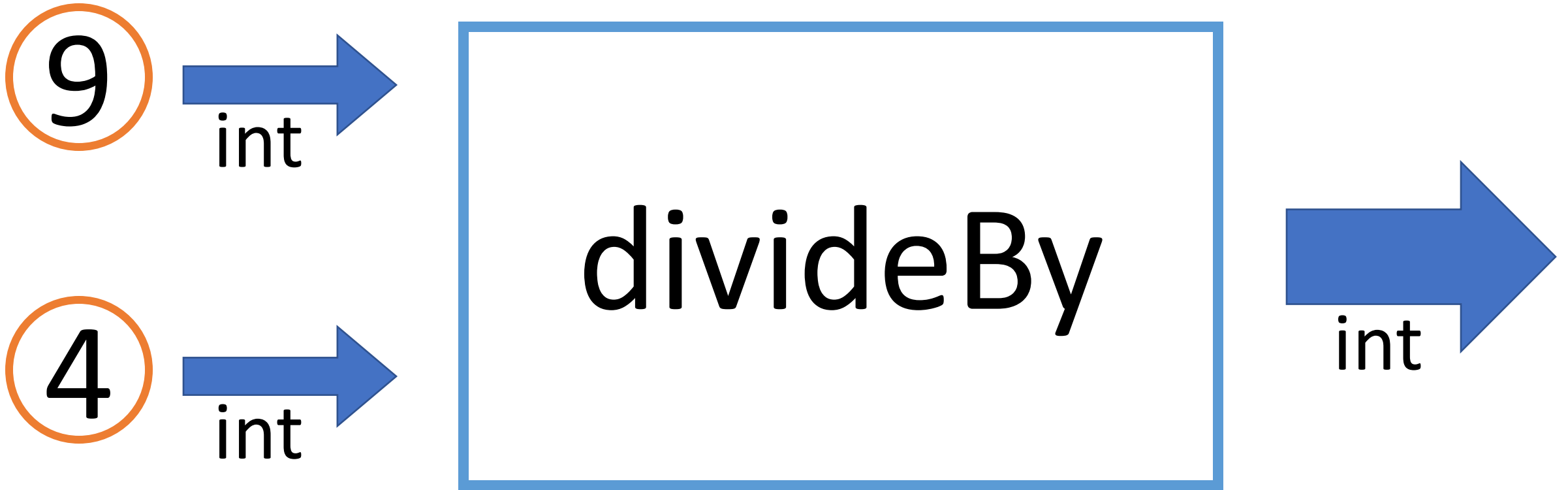
Parametrización de Datos



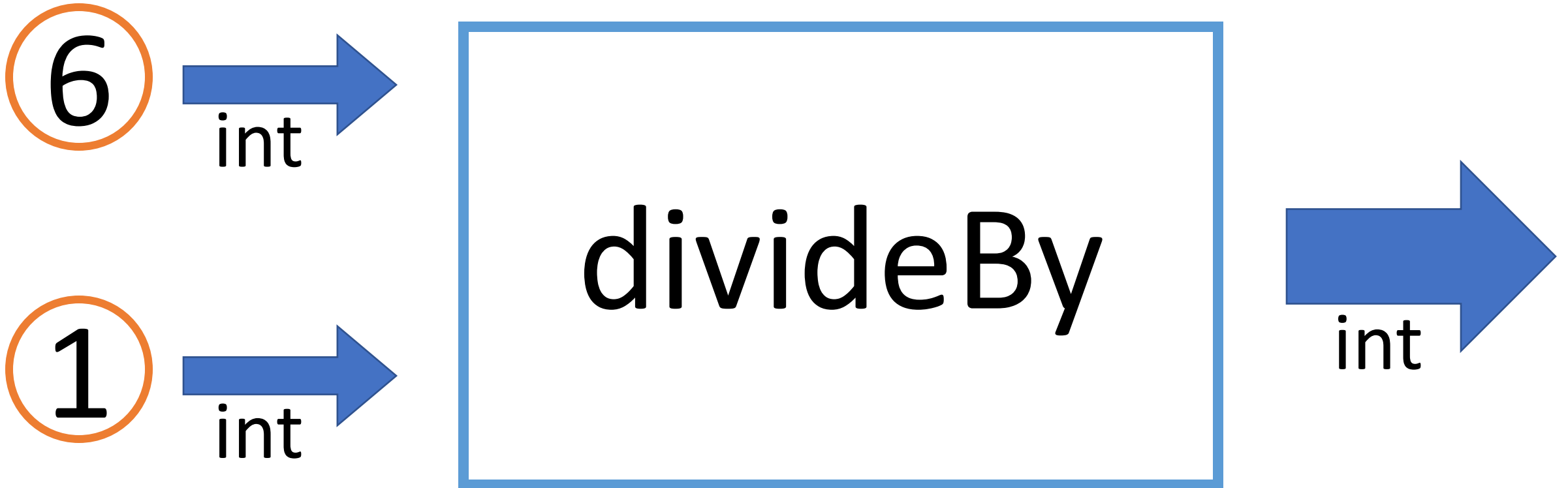
Parametrización de Datos



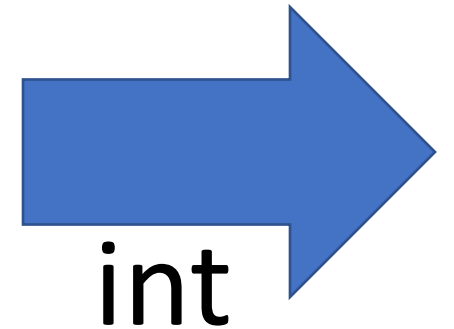
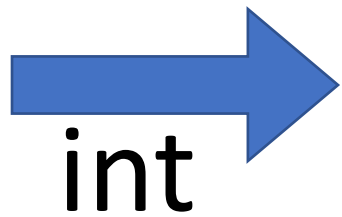
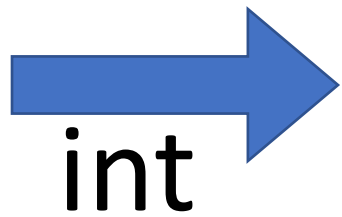
Parametrización de Datos



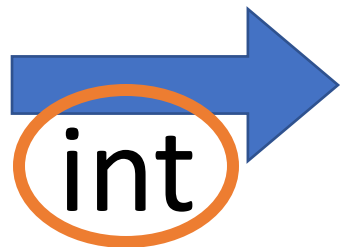
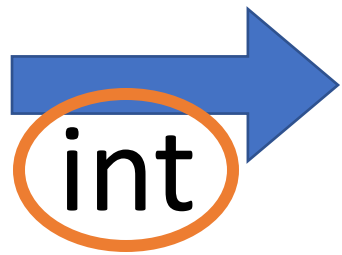
Parametrización de Datos



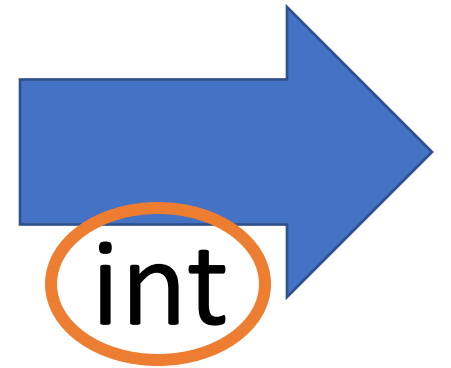
Pero...



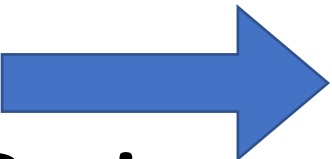
Pero...




aMethod



Pero...

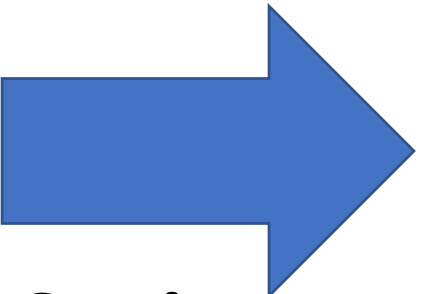


String




String

aMethod




String

Pero...

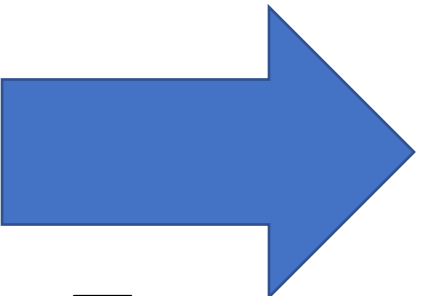


aType



aType

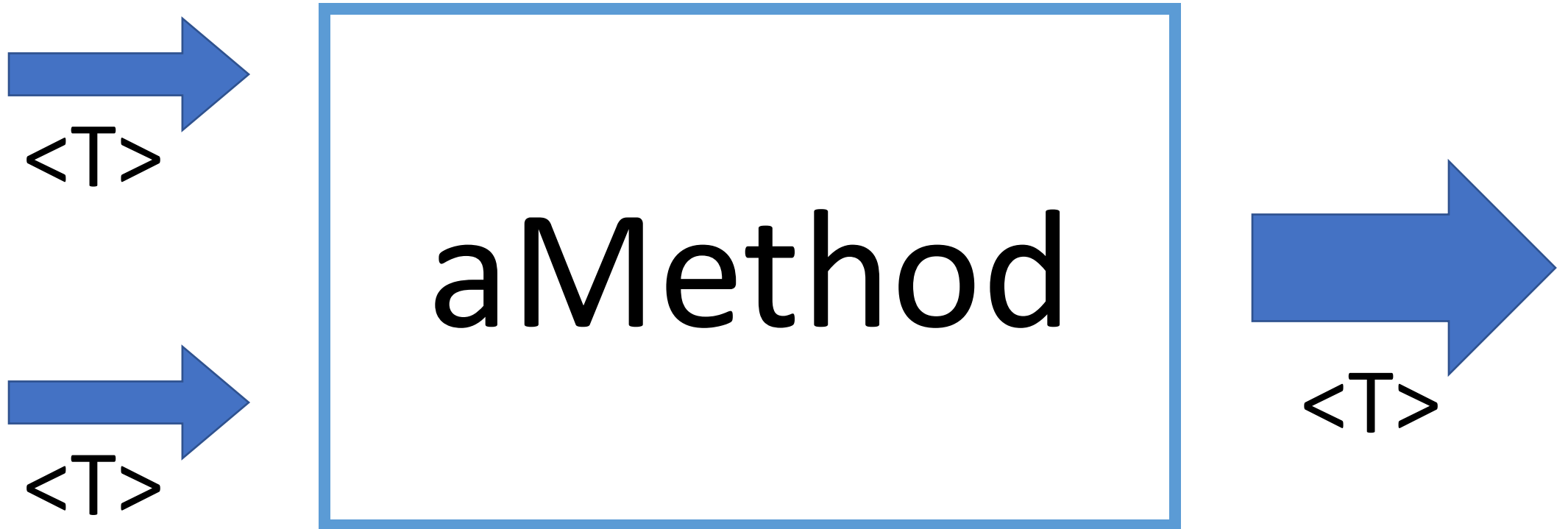
aMethod



aType

¿Cómo lo logramos?

Usando tipos de datos genéricos



Tipos de Datos Genéricos

Parametrización de Tipos

Permiten crear clases, interfaces y métodos en los que **los tipos de datos** sobre los que opera se especifican como parámetros.

En una **parametrización de tipos** usamos parámetros de tipo.

Así podemos crear, clases genéricas, interfaces genéricas, o métodos **genéricos**.

Ventajas

El **código genérico** trabaja automáticamente con el tipo de datos pasados a su parámetro de tipo.


Ejemplo: La búsqueda binaria es *la misma* ya sea que se esté buscando números, o cadenas de caracteres, o estudiantes, o materias, etc.

Con genéricos, se puede definir el algoritmo una vez y aplicarlo a varios tipos de datos sin mayor esfuerzo adicional.

Genéricos en Java

Cómo se hace?

Hay dos formas de usar tipos de datos genéricos en Java:

- Usando la clase **Object** 
- Usando parámetros de tipo

La clase Object

Es la primera forma de usar genéricos en Java



Debido a que Object es la superclase de toda clase, una referencia de Object puede referirse a cualquier tipo de objeto.

Object aMethod(**Object** o)



aMethod(3)

aMethod("Hola")

aMethod(1.78)

aMethod(aCar)

Pero...



```
Object o = aMethod(new Student(...));
```

```
String theResult = (String) o;
```

```
int theResult = (int) o;
```

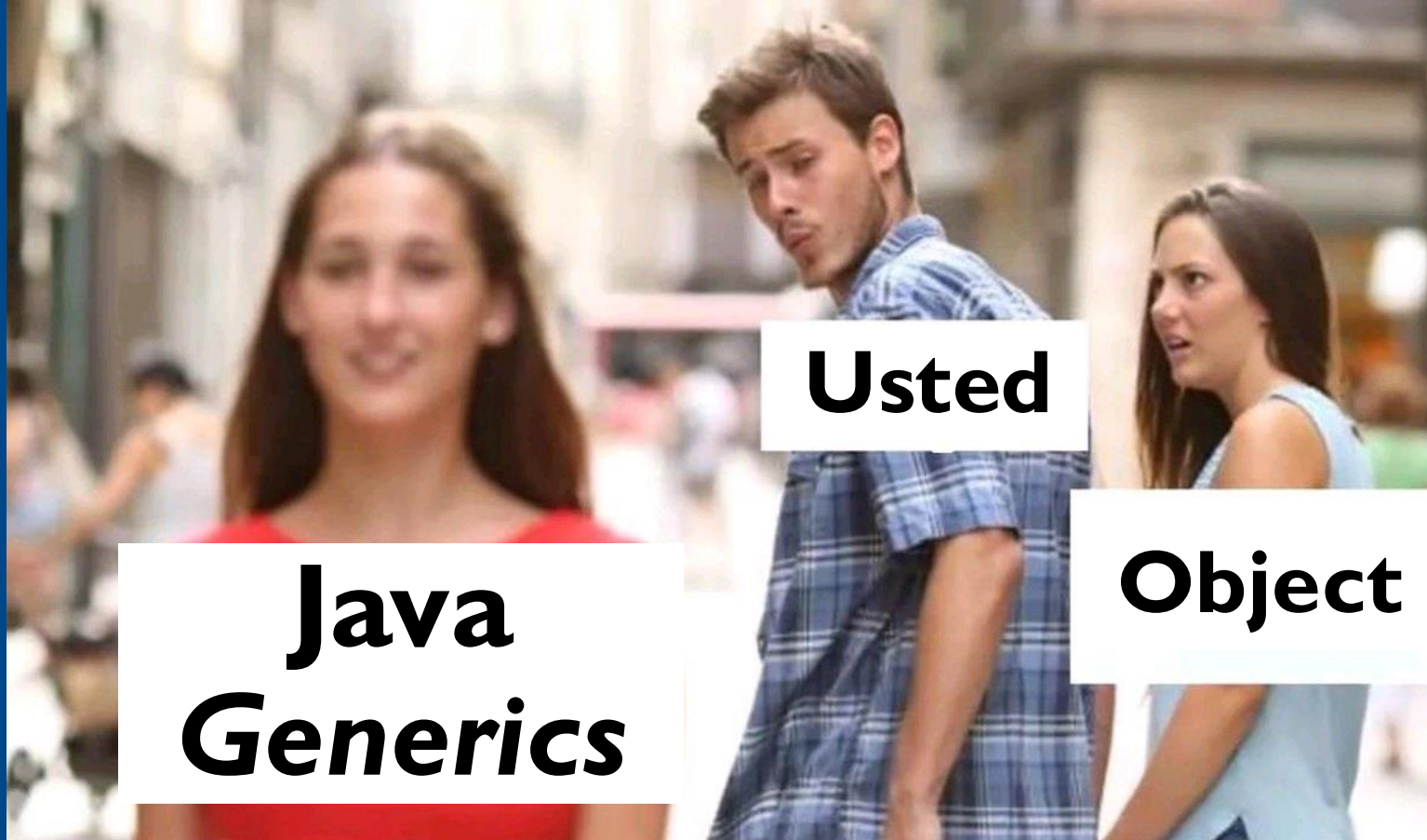
¿Cómo se hace?

Casts necesarios para convertir explícitamente de **Object** al tipo real de datos sobre los que se va a operar.



No es seguro: Posibles errores durante tiempo de ejecución cuando los casts no son posibles.

En 2004, Java introdujo *Generics* para solucionar este problema y lograr seguridad en tiempo de compilación.



**Java
Generics**

Usted

Object

Scenario

Objetivo: Diseñar una clase que permita almacenar un artículo en una caja fuerte



article

Box

Una posible implementación

```
public class Box {  
  
    private String article = null;  
  
    public Box(String article) {  
        this.article = article;  
    }  
  
    public String getArticle() {  
        return article;  
    }  
  
    public void setArticle(String article) {  
        this.article = article;  
    }  
  
}
```

Una posible implementación

```
public class Box {  
    private String article = null;  
  
    public Box(String article) {  
        this.article = article;  
    }  
  
    public String getArticle() {  
        return article;  
    }  
  
    public void setArticle(String article) {  
        this.article = article;  
    }  
}
```

Una posible implementación

```
public class Box {  
  
    private String article = null;  
  
    public Box(String article) {  
        this.article = article;  
    }  
  
    public String getArticle() {  
        return article;  
    }  
  
    public void setArticle(String article) {  
        this.article = article;  
    }  
  
}
```

Uso

```
public class Main {  
    public static void main(String[] args) {  
        Box box1 = new Box("gold bar");  
        Box box2 = new Box("crown");  
        System.out.println(box1.getArticle());  
        System.out.println(box2.getArticle());  
    }  
}
```

Pero...

Solución limitada: artículos representados solo como objetos **String**

Requerimiento adicional: Se necesita almacenar más información acerca de los artículos, permitiendo cualquier TDA.

Por ejemplo:

- TDA DiscoDuro

- TDA Joya

- TDA Laptop

- TDA Dinero

- TDA Camara

Usando Object

```
public class Box {  
  
    private Object article = null;  
  
    public Box(Object article) {  
        this.article = article;  
    }  
  
    public Object getArticle() {  
        return article;  
    }  
  
    public void setArticle(Object article) {  
        this.article = article;  
    }  
  
}
```



Usando Object

```
public class Box {  
    private Object article = null;  
  
    public Box(Object article) {  
        this.article = article;  
    }  
  
    public Object getArticle() {  
        return article;  
    }  
  
    public void setArticle(Object article) {  
        this.article = article;  
    }  
}
```



Usando Object

```
public class Box {  
  
    private Object article = null;  
  
    public Box(Object article) {  
        this.article = article;  
    }  
  
    public Object getArticle() {  
        return article;  
    }  
  
    public void setArticle(Object article) {  
        this.article = article;  
    }  
  
}
```



Usando Object



```
public class Main {  
    public static void main(String[] args) {  
        Box box1 = new Box("gold bar");  
        String article = box1.getArticle();  
    }  
}
```

Usando Object



```
public class Main {  
    public static void main(String[] args) {  
        Box box1 = new Box("gold bar");  
        String article = box1.getArticle();  
    }  
}
```

Usando Object



```
public class Main {  
    public static void main(String[] args) {  
        Box box1 = new Box("gold bar");  
        String article = (String) box1.getArticle();  
    }  
}
```

Usando Object



```
public class Main {  
    public static void main(String[] args) {  
        Box box1 = new Box("gold bar");  
        String article = (String) box1.getArticle();  
    }  
}
```

¿Cuál es el problema?

```
public class Box {  
    private Object article = null;  
  
    public Box(Object article) {  
        this.article = article;  
    }  
  
    public Object getArticle() {  
        return article;  
    }  
  
    public void setArticle(Object article) {  
        this.article = article;  
    }  
}
```



Generalizando

```
public class Box<T> {  
  
    private T article = null;  
  
    public Box(T article) {  
        this.article = article;  
    }  
  
    public T getArticle() {  
        return article;  
    }  
  
    public void setArticle(T article) {  
        this.article = article;  
    }  
  
}
```


<T>

```
public class Box<T> {  
    private T article = null;  
  
    public Box(T article) {  
        this.article = article;  
    }  
  
    public T getArticle() {  
        return article;  
    }  
  
    public void setArticle(T article) {  
        this.article = article;  
    }  
}
```

<T>

```
public class Box<T> {  
    private T article = null;  
  
    public Box(T article) {  
        this.article = article;  
    }  
  
    public T getArticle() {  
        return article;  
    }  
  
    public void setArticle(T article) {  
        this.article = article;  
    }  
}
```

T es un parámetro
de tipo

(siempre entre <>)

Box es una clase
genérica

Usando la clase genérica Box

```
public class Main {  
    public static void main(String[] args) {  
        Box<String> box1 = new Box("Joya");  
        Box<Integer> box2 = new Box(1000);  
        Box<Student> box3 = new Box(new Student ("Gonzalo", "Méndez"));  
        System.out.println(box1.getArticle());  
        System.out.println(box2.getArticle());  
        System.out.println(box3.getArticle());  
    }  
}
```

Algunas Convenciones

- Los nombres de parámetros de tipo son:
 - Simples
 - Letras mayúsculas
- Nombres de tipos de parámetros más comúnmente usados
 - E - Element (usado ampliamente por el Framework de Colecciones Java)
 - K - Key
 - N - Number
 - T - Type
 - V - Value
 - S,U,V etc. - 2nd, 3rd, 4th types

Otros Puntos Importantes

1. Los genéricos funcionan solo con tipos compuestos

```
Box<int> b1 = new Box(28); // ERROR  
Box<char> b2 = new Box('a'); // ERROR
```

Pero sí se puede usar clases envoltorias:

```
Box<Integer> b1 = new Box(28); // OK  
Box<Character> b2 = new Box('a'); // OK
```

2. Instancias de una clase generica *pueden* diferir en tipo

```
public class Main {  
    public static void main(String[] args) {  
        Box<String> box1 = new Box("Joya");  
        Box<Integer> box2 = new Box(1000);  
        box1 = box2;  
    }  
}
```

incompatible types: Box<Integer> cannot be converted to Box<String>

(Alt-Enter shows hints)

Box<String> != Box<Integer>

3. El parámetro de tipo puede ser omitido

```
public class Main {  
    public static void main(String[] args) {  
        Box box3 = new Box("Joya");  
        String article2 = box3.getArticle();  
        Object article1 = box3.getArticle();  
        String article3 = (String) box3.getArticle();  
    }  
}
```

incompatible types: Object cannot be converted to String

(Alt-Enter shows hints)

4. Parametrización NO es overloading

Overloading permite a una clase tener más de un método con el mismo nombre pero con una lista de argumentos diferentes.

```
static int add(int a, int b)  
static String add(String a, String b)
```

Parametrización permite *variar* el tipo de dato que un método recibirá

```
<T> add(<T> a, <T> b)
```

Todas las versiones de un método overloaded retornan el mismo tipo de dato

Ejercicio

Crear una clase que permita representar la **relación** entre dos entidades cualesquiera. A continuación se muestran ejemplos de las relaciones que se quiere representar; la descripción de la relación entre las entidades aparece subrayada:

Este cachorro es mascota de esta señora

Este empresario posee este conjunto de empresas

Esta ciudad es la capital de este país

Este veterinario atiende a este cachorro

Este avión se dirige a esta ciudad

```
public class Relation<T, R> {

    private T entitiy1;
    private R entitiy2;
    private String description;

    public Relation(T entitiy1, R entitiy2, String description) {
        this.entitiy1 = entitiy1;
        this.entitiy2 = entitiy2;
        this.description = description;
    }

    public T getEntitiy1() {
        return entitiy1;
    }

    public void setEntitiy1(T entitiy1) {
        this.entitiy1 = entitiy1;
    }

    public R getEntitiy2() {
        return entitiy2;
    }

    public void setEntitiy2(R entitiy2) {
        this.entitiy2 = entitiy2;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```