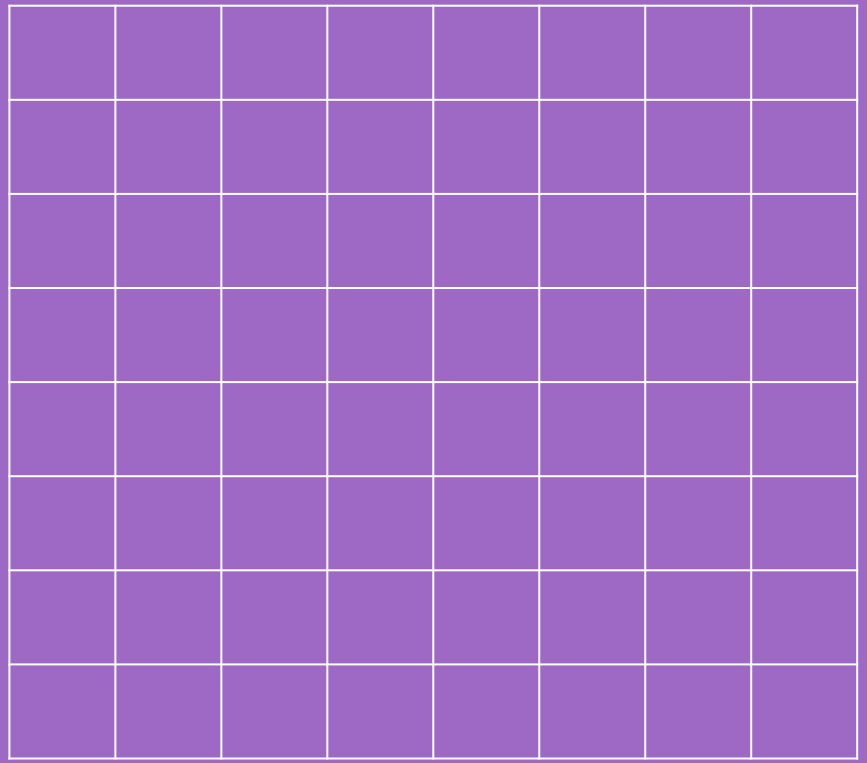
Argentina
programa
4.0

# Excepciones y Colecciones en Java

"Desarrollador Java Inicial"

# Excepciones



# Tipos de errores

Argentina programa 4.0

¿ Qué tipos de errores pueden ocurrir durante la ejecución de un programa / aplicación?

- Situación inesperada
- Error de negocio / Flujos alternativos

En Java ambos se implementan a través de excepciones o Exceptions

Situaciones que tengo en cuenta durante el desarrollo:

- Login incorrecto
- Datos en formato incorrecto
- Modificar los datos de una persona dada de baja
- El archivo necesario no existe
- El mensaje no puede llegar a destino
- etc...

# Códigos de error

Argentina programa 4.0

Primero vamos a ver las siguientes implementaciones

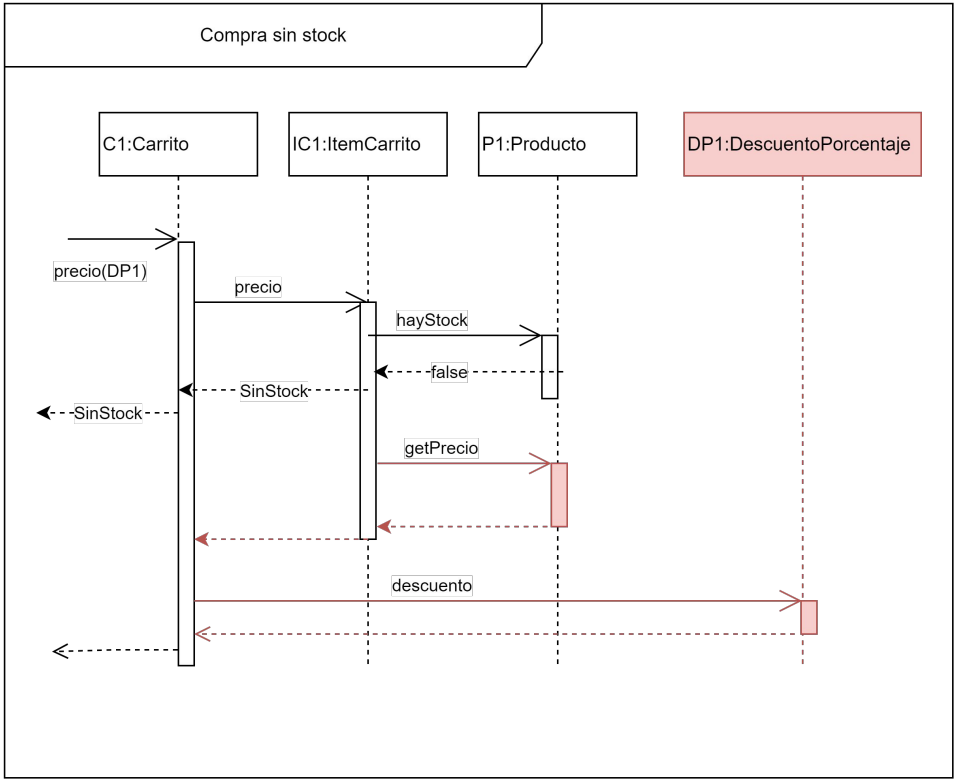
- Login#ingresar(usuario,password): int -> 1 Login ok 2 login error
- RepoUsuario#modificar(usuario):String -> ok,  usuario no existe, nombre en formato incorrecto, apellido vacío ,  ... etc

Este es un mecanismo indicado para cuestiones sencillas (por ejemplo en la mayoría de los SO un proceso que termina correctamente retorna cero y si hay algún error cualquier otro número), pero cuando las situaciones de error son más complejas son poco adecuadas

# Excepciones

Mecanismo utilizado por diversos lenguajes de programación para generar y manejar los errores durante la ejecución de un proceso.

En este ejemplo, lo rojo nunca se ejecuta y el programa se corta ante la excepción de SinStockException



# Implementación

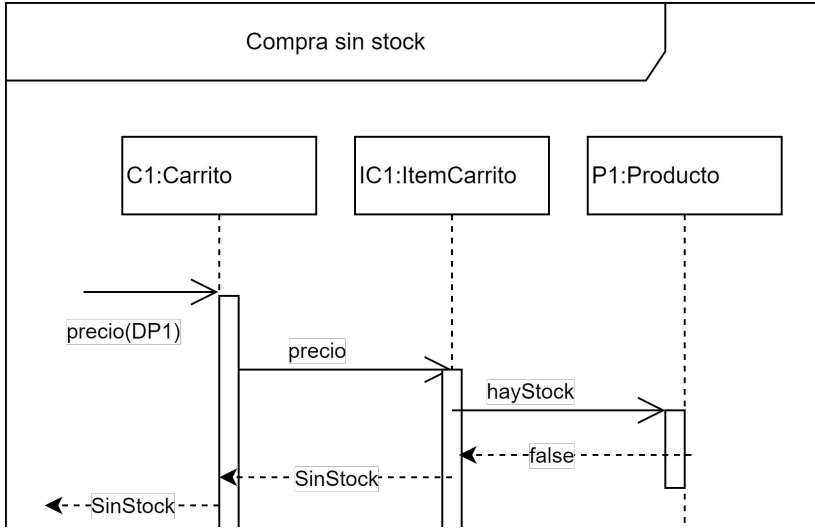
Argentina programa 4.0

1. Crear una clase que herede de Exception o RuntimeException

```
public class NoHayStockException extends Exception {
      private final Producto producto;
      public NoHayStockException(Producto producto) {
            this.producto = producto;
      }}
```

2. Arrojar la Excepción

```
public class ItemCarrito {
      public float precio() throws NoHayStockException {
            if (!producto.hayStock()) {
                  throw new NoHayStockException(producto);
            }
            return this.cantidad * this.producto.getPrecio();
      }}
```



# Implementación

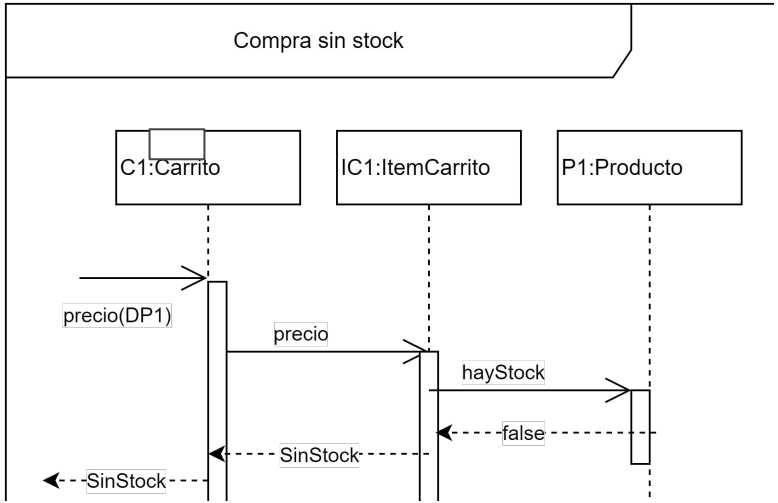
Argentina programa 4.0

## 3. “Subir” la Exception

```
public class CarritoCompra {
            public float precio() throws
      NoHayStockException {
                  return item1.precio() + item2.precio()
            + item3.precio();
            }
}
```

## 4. Tratar la excepción

```
public static void main(String[] args) {
      CarritoCompra carrito = new CarritoCompra();
      // Se arma el carrito …
      try {
            Float precio = carrito.precio();
            System.out.println("el precio del carrito es: "+ precio.toString());
      } catch (NoHayStockException e) {
            System.out.println("No hay stock de al menos uno de los productos");
      }
}
```



# Excepciones Comunes

Argentina programa 4.0

Java y algunas librerías base, ya establecieron excepciones comunes y algunas de ellas deberían ser utilizadas o heredadas antes de crear otras.

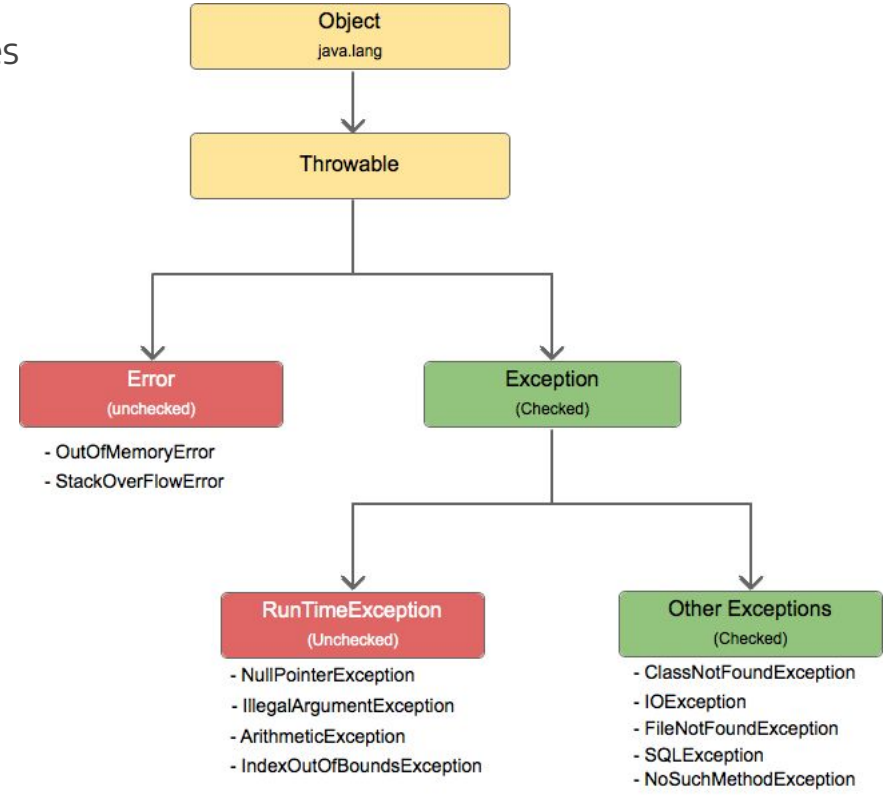
No chequeadas

- NullPointerException
- DivisionByCeroException
- IllegalStateException

Chequeadas

- FileNotFoundException
- IOException

....



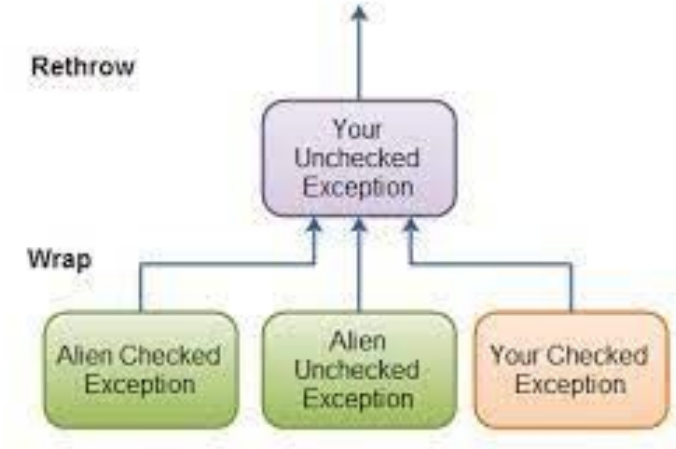
# Para tener en cuenta

Argentina programa 4.0

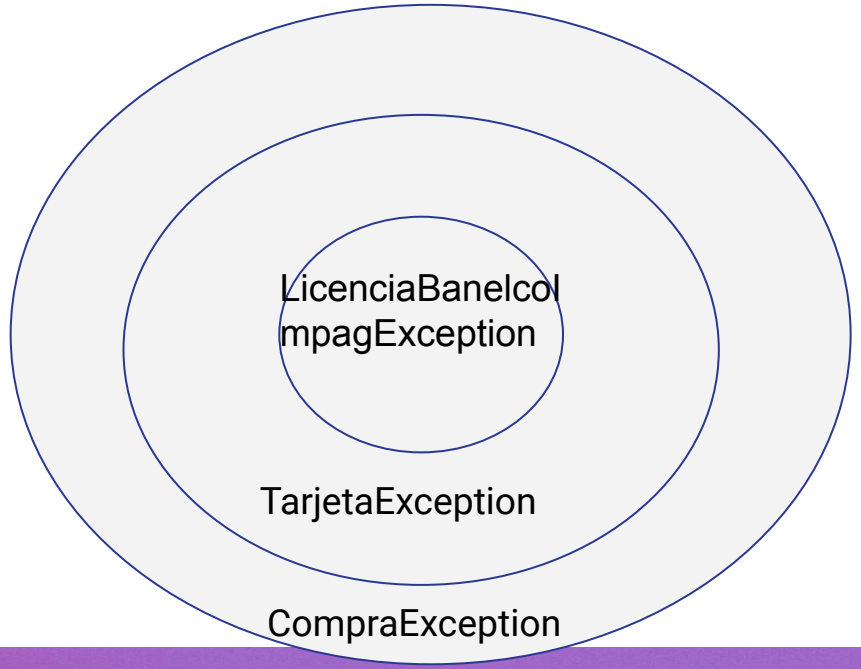
- No siempre es fácil determinar cuándo crear una excepción
- Depende un poco de mi marco de referencia, pero en general hay que pensar que errores un “usuario” puede llegar a cometer (olvidarse de la UI)
- ¿Qué tan específico ser con las excepciones?
    - Mientras más preciso se es, más fácil es saber QUÉ SALIÓ MAL
    - También tener en cuenta que las excepciones pueden tener datos asociados
- ¿Cuántas Crear?
    - Tampoco sirve tener  muchos tipos de error...
    - No se puede tener en cuenta todo

# Empaquetado de excepciones

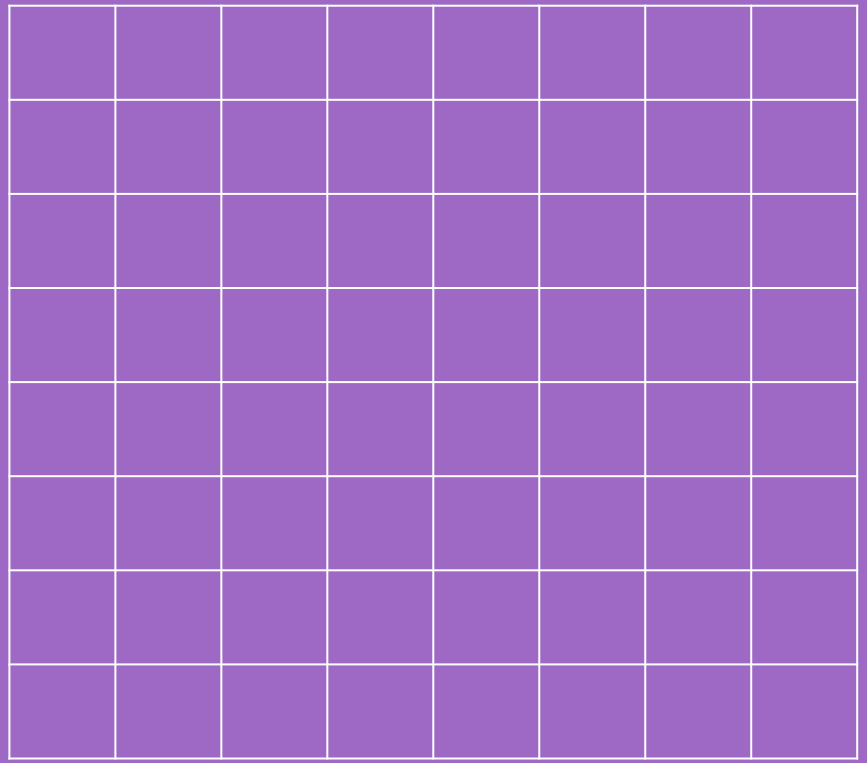
Una práctica muy común es empaquetar una excepción dentro de otra. Esto genera una pila de excepciones, donde cada “dominio” o parte del programa le llega el error específico y si quiere saber la causa raíz tiene la información suficiente.




comprar



# Colecciones



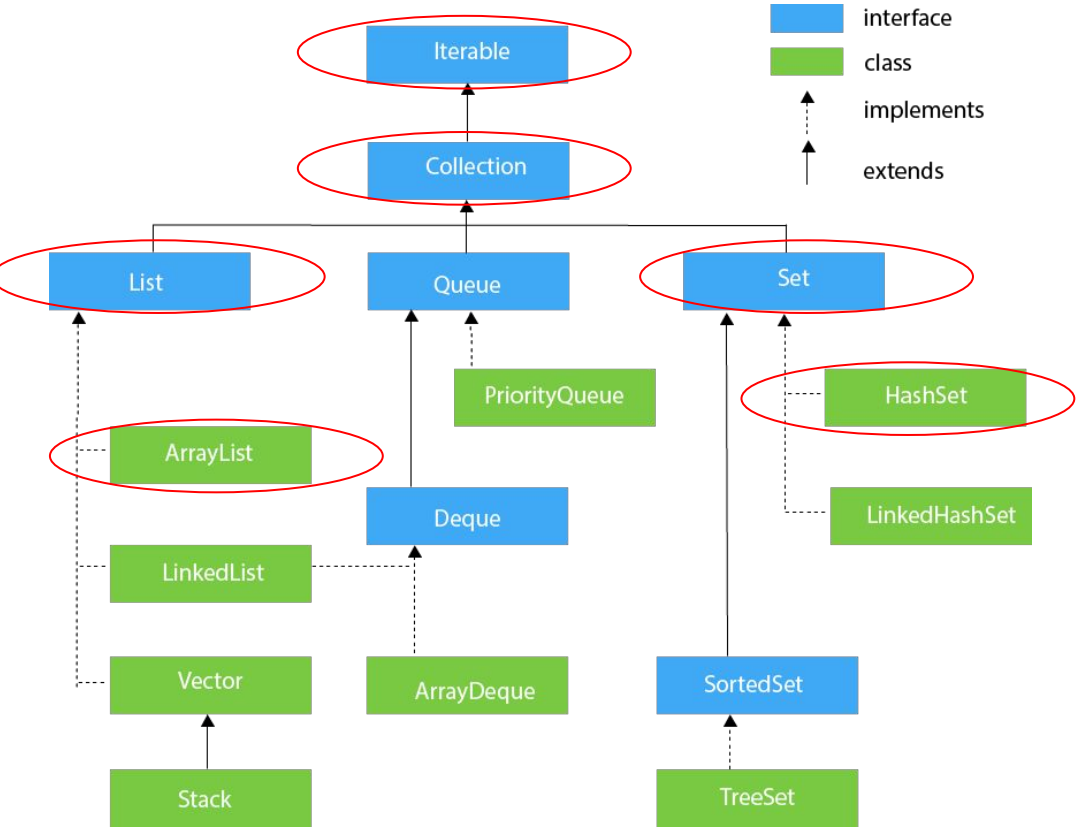
# Colecciones

Argentina programa 4.0

Las colecciones son grupos de objetos, que a diferencia de los arrays [] que vimos antes, pueden variar su tamaño dinámicamente.

Debido a que existen numerosos problemas asociados a generar estas agrupaciones, hay una jerarquía no tan simple para representar estos problemas.

# Colecciones - Jerarquía




Iterable -> hasNext / next
Collection -> add / remove / contains
List -> add(int index, E element) / get (int index)
Set → no permite repetidos
Varían en la representación interna de los datos, por ende van tener distintos tiempos de acceso, lectura , escritura según los datos que tengan.
En este curso, nos enfocaremos en 3 implementaciones:

- ArrayList
- HashMap
- HashSet

https://www.javatpoint.com/collections-in-java

# Colecciones - Aplicación de una Lista

Argentina programa 4.0

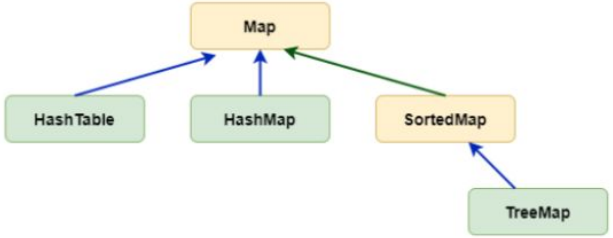
```
import java.util.List;
import java.util.ArrayList;

class CarritoCompra {

      private List<ItemCarrito> items;

      public CarritoCompra() {
            this.items = new
            ArrayList<ItemCarrito>();
      }
      public void agregarItem(ItemCarrito ic) {
            this.items.add(ic);
      }
      public void quitarItem(ItemCarrito ic) {
            this.items.remove(ic);
      }
      public List<ItemCarrito> getItems() {
            return items;
      }
```

En este caso, la colección se declara como atributo y es necesaria usar la sintaxis <> para indicar de qué tipo son los elementos de la lista. Notar por otro lado que cuando se declara, se hace con la interfaz y no con la implementación

Es necesario construir la colección, en este caso el constructor del objeto es un buen lugar

Se agregan métodos públicos para modificar la lista y obtener sus valores

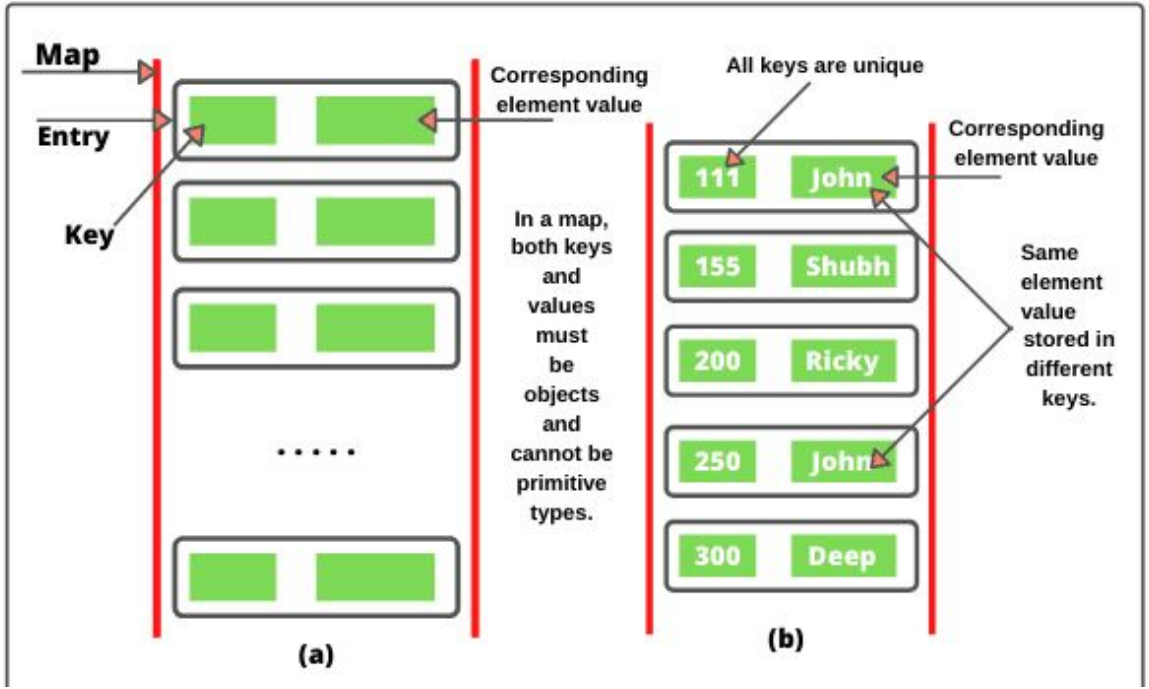
# Colecciones - Map




Map

- put(key,value)
- get (key) : value
- hasKey : boolean
- keySet : Set

*Tener cuidado porque no es el mismo concepto con los map de otros lenguajes...*




https://www.scientecheasy.com/2020/10/map-in-java.html/

# Sin repetidos

Argentina programa 4.0

HashSet

HashMap

Si no admiten repetidos ¿cómo sabe si le están agregando un elemento que ya tiene?

`Object#hashcode-> número “único” - identidad`

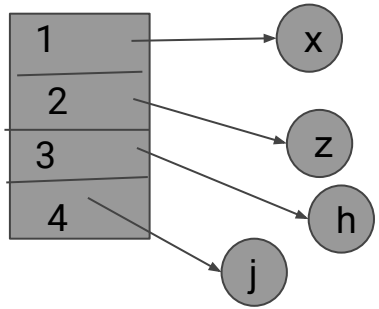
Puedo/Debo sobrescribir este método, por ejemplo utilizando los atributos

```
import java.util.Objects;
[...]
@Override
public int hashCode() {
 return Objects.hash(nombre, descripcion, fechaAlta, pesoKg, precio, stock);
}
```

# Consideraciones en el código


¿Qué consideraciones cree que hay que tener ahora que tenemos una colección?, por ejemplo:

1. Cuando agregamos un elemento: Carrito#addItem(ItemCarrito)
2. Si tengo una implementación de getItems()




1) Inicialización

```
public CarritoCompra() {
    this.items = new ArrayList<>();
}
```

1) Clonado

```
public Collection<ItemCarrito> getItems() {
    Collection<ItemCarrito> items2 = new ArrayList<>();
    items2.addAll(this.items);
    return items2;
}
```

# Colecciones - Uso - Iteraciones

Argentina programa 4.0

```
float precioFinal = 0;
for (ItemCarrito item: this.items ) {
     float precioItem = item.precio();
     precioFinal += precioItem ;
}
//----------------------------------------
Iterator<ItemCarrito> iterator = this.items.iterator();
while (iterator.hasNext()){
     ItemCarrito item = iterator.next();
     float precioItem = item.precio();
     precioFinal += precioItem ;
}
//----------------------------------------
precioFinal = this.items.stream().mapToDouble(
     item -> item.precio()).map(precio -> precio ).sum();
```

https://www.scientecheasy.com/2020/10/map-in-java.html/

# Referencias

Argentina programa 4.0

- https://www.manishsanger.com/java-exception-hierarchy/
- https://stackify.com/best-practices-exceptions-java/
- https://www.jrebel.com/blog/java-collections-cheat-sheet
- https://www.jrebel.com/blog/java-generics-cheat-sheet
- https://www.jrebel.com/blog/java-8-cheat-sheet
- https://www.jrebel.com/blog/java-streams-cheat-sheet

Argentina
programa
4.0

# Gracias!