



Définition La généricité permet de paramétrer du code avec des types de données Exemple: Classe ArrayList<T> dont le code est paramétré par un type T Pour l'utiliser il faut passer un type en argument : new ArrayList<Employe>()



Avant le JDK 5 Les éléments des collections étaient déclarés de type Object Impossible de déclarer une collection d'Employe ou une collection de Livre Classe ArrayList: boolean add(Object o) Object get(int i)

```
Conséquences

Des erreurs ne sont repérées qu'à l'exécution et pas à la compilation

Il faut sans arrêt caster les éléments des collections

Le plus grave ?

Richard Grin Généricité page 8
```



Vocabulaire ArrayList<E> est un type générique E est un paramètre de type E sera remplacé par un argument de type: ArrayList<Integer> 1 = new ArrayList<Integer>(); ArrayList<Integer> est une instanciation du type générique; c'est un type paramétré Types « raw » : ancien type non générique ArrayList Richard Grin Généricité page 10

```
List<Employe> employes =
    new ArrayList<Employe>();
Employe e = new Employe("Dupond");
employes.add(e);
// On ajoute d'autres employés
. . .
for (int i = 0; i < employes.size(); i++) {
    System.out.println(
        employes.get(i).getNom());
}
Plus besoin de cast; pourquoi?</pre>
```

```
Autre exemple d'utilisation

List<Employe> employes =
    new ArrayList<Employe>();
Employe e = new Employe("Dupond");
employes.add(e);
// On ajoute d'autres employés
. . .
// Ajoute un livre au milieu des employés
Livre livre = new Livre(...);
employes.add(livre);

Ca compile ?
Ca s'exécute ?
```

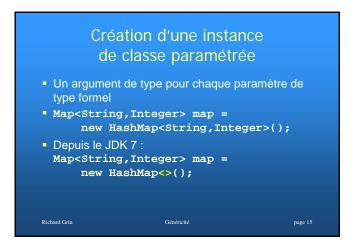
```
Utilisations des paramètres de type

Peuvent être utilisés pour déclarer des variables, des paramètres, des tableaux ou des types retour de méthodes:

E element;
E[] elements;

Ne peuvent pas être utilisés pour
créer des objets ou des tableaux
comme super-type d'un autre type

new E[]
new E[]
```





Instanciation d'une méthode générique On peut appeler une méthode paramétrée en la préfixant par un argument de type : <string>m() Mais le plus souvent on peut omettre ce préfixe car le compilateur peut « deviner » le type d'après le contexte

```
Inférence de type

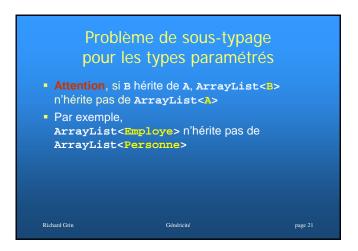
ArrayList<Personne> liste;
...
Employe[] res =
    liste.toArray(new Employe[0]);

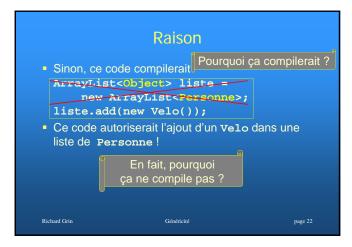
Rappel:<T> T[] toArray(T[] a)
    Que « devine » le compilateur pour T?

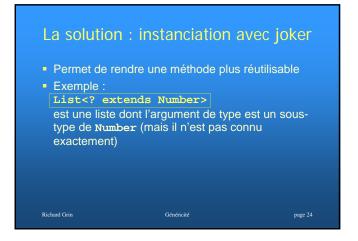
Le compilateur infère
liste.<Employe>toArray(new Employe[0]);

Richard Grin Génériché page 19
```









```
Imploye hérite de Personne
Une bonne méthode pour afficher
les noms d'une liste de personnes

public static void afficheNoms(
    List<? extends Personne> liste) {
    for(Personne p : liste) {
        System.out.println(p.getNom());
    }
}

Pourquoi ça compile ?

List<Employe> liste =
    new ArrayList<Employe>();
...
afficheNoms(liste); Pourquoi ça compile ?
Richard Grin Génériché page 25
```

Les « types » avec joker - <?> désigne un type inconnu - <? extends A> désigne un type inconnu qui est un sous-type de A (A compris) - <? super A> désigne un type inconnu qui est un sur-type de A (A compris) - A peut être un type quelconque, sauf un type primitif - On dit que A contraint le joker

Où peuvent apparaître les « types » avec joker ? • Attention, abus de langage! ce ne sont pas des vrais types • Seulement pour instancier un type paramétré: List<? extends Number> • Ne peuvent pas typer une variable: <? extends Number> n;

```
Où peuvent apparaître les instanciations de types avec joker?

Dans une classe quelconque (générique ou non)

Pour typer des variables, des paramètres, des tableaux ou les valeurs retour des méthodes:

List<? extends Number> 1;

Ne peuvent pas être utilisées pour créer des objets ou des tableaux

Ne peuvent pas être des super types:

class C implements Comparable<? super C>
```

```
static <T> void
fill(List<? super T> liste, T elem) {
  int size = liste.size();
  for (int i = 0; i < size; i++)
      liste.set(i, elem);
}

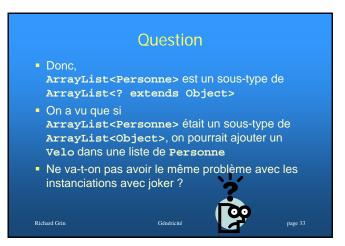
Remplace tous les éléments d'une liste par un certain
élément
Intuition : pour remplir avec un objet de type T, le type
des éléments de la liste doit être un sur-type de T

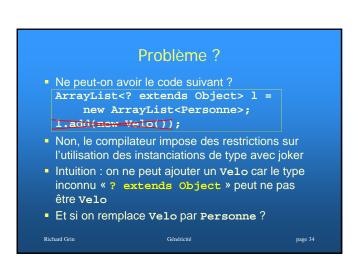
Richard Grin Généricité page 30</pre>
```

static <T> void fill(List<? super T> liste, T elem) { int size = liste.size(); for (int i = 0; i < size; i++) liste.set(i, elem); Possible de remplacer par une boucle for-each? Remplace tous les éléments d'une liste par un certain élément Intuition : pour remplir avec un objet de type T, le type</pre>

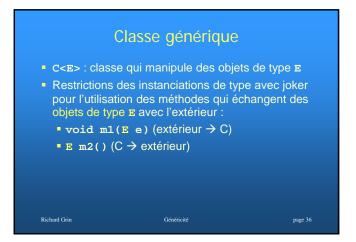
des éléments de la liste doit être un sur-type de T

Sous-typage Soit B une classe fille de A List est-il un sous-type de List<? extends A>? Oui, car B est bien un sous-type de A List<? extends A> est-il un sous-type de List? Non, le « ? » peut représenter un sous-type de A qui n'est ni B, ni un sous-type de B

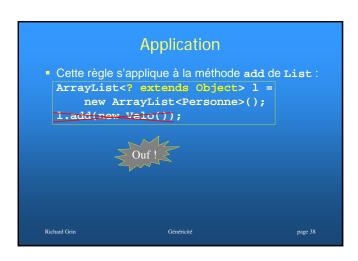




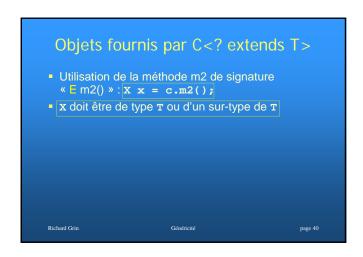
Échanges d'une classe avec l'extérieur Une classe fournit des objets à l'extérieur par ses méthodes qui renvoient des objets: TypeRetour m2() Elle reçoit des objets de l'extérieur par ses méthodes qui prennent des objets en paramètre: Ret m1(TypeDuParametre x)

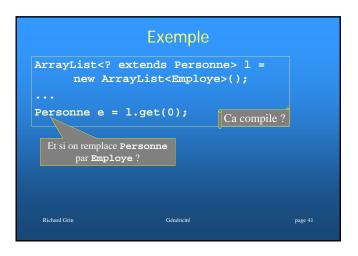


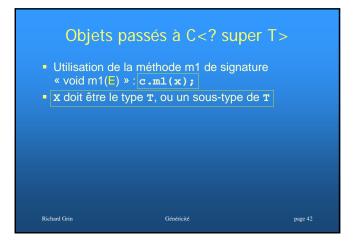
Objets passés à C<? extends T> ■ Utilisation de la méthode m1 de signature « void m1(E) » : c.ml(x); ■ « Signature » de m1 pour cette instantiation : « void m1(? extends T) » ■ Comment doit être le type x de x pour que l'affectation « x → ? extends T » soit acceptée par le compilateur ? ■ Aucun x pour lequel cette affectation est assurée de ne pas poser de problème ■ Donc l'appel d'une telle méthode est interdite Richard Grin Généricité page 37



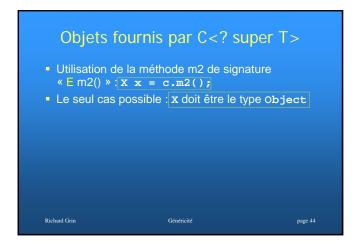
Autres restrictions • Il existe d'autres restrictions pour toutes les instanciations avec joker Richard Grin Genéricité page 39

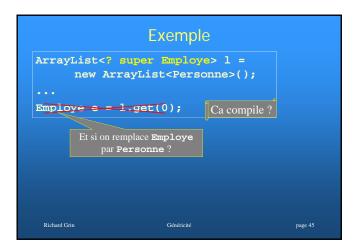


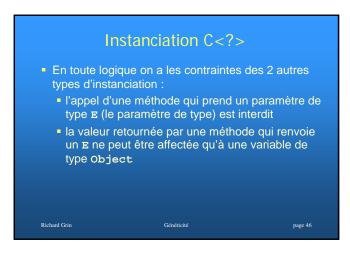




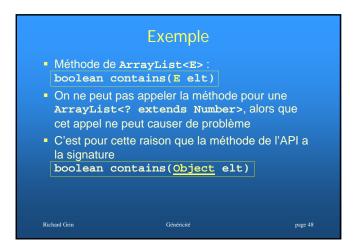
Exercice • void f(List<? super Cercle> liste) { liste.add(machin); De quel type peut être machin? • machin doit être déclaré du type Cercle (ou d'un sous-type) Richard Grin Généricité page 43



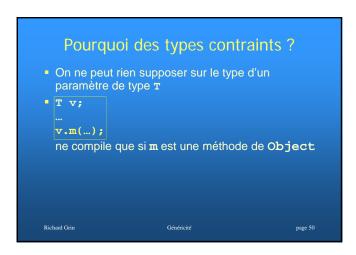




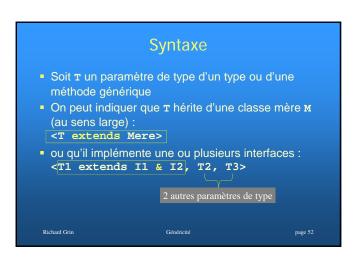
Utilité de ces règles Ces règles sont nécessaires pour obtenir du code correct On l'a vu pour la méthode add(E elt) de la classe ArrayList<E> Quelquefois, cependant, ces règles ajoutent une contrainte qui ne sert à rien C'est en particulier le cas pour les méthodes qui ne modifient pas la collection



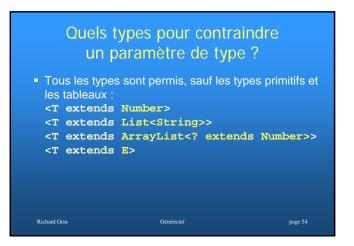
Types paramétrés contraints (bounded en anglais)



Problème Cependant, pour écrire une méthode de tri « sort (ArrayList<E>) », il est indispensable de pouvoir comparer les éléments de la liste Il faut un moyen de dire que E contient une méthode pour comparer les instances du type Par exemple en disant que E implémente Comparable



Syntaxe (2) Si T hérite d'une classe mère et implémente des interfaces, la classe mère doit apparaître en premier: T extends Mere & I1 & I2>



```
public static <T extends Comparable<? super T>>
T min(List<T> liste) {
  if (liste.isEmpty())
    return null;
T min = liste.get(0);
  for (int i = 1; i < liste.size(); i++) {
    if (liste.get(i).compareTo(min) < 0)
       min = liste.get(i);
    }
  return min;
}</pre>
Richard Grin Génériché page 55
```

```
public static <T extends Comparable<? super T>>
T min(List<? extends T> liste) {
  if (liste.isEmpty())
    return null;
T min = liste.get(0);
  for (int i = 1; i < liste.size(); i++) {
    if (liste.get(i).compareTo(min) < 0)
       min = liste.get(i);
  }
  return min;
}</pre>
Richard Grin Généricié page 56
```

```
public static <T extends Comparable<? super T>>
T min(Collection<? extends T> coll) {
  if (coll.isEmpty())
    return null;
    T min = coll.get(0);
  for (int i = 1; i < coll.size(); i++) {
    if (coll.get(i).compareTo(min) < 0)
        min = coll.get(i);
    }
    return min;
}

Richard Grin Genericit page 57</pre>
```

```
public static <T extends Comparable<? super T>>
T min(Collection<? extends T> coll) {
  if (coll.isEmpty()) return null;
  Iterator<? extends T> it = coll.iterator();
  T min = it.next();
  while (it.hasNext()) {
    T next = it.next();
    if (next.compareTo(min) < 0)
        min = next;
  }
  return min;
    Utiliser un itérateur
  plutôt que get</pre>
Richard Grin Généricité page 58
```

Implémentation de la généricité et restrictions associées

Instanciation des classes génériques ArayList<Integer> et ArrayList<Employe> sont représentés dans la JVM par la seule classe ArrayList Richard Grin Générité page 60

Une contrainte importante Dans le cahier des charges de la généricité en Java : tout le code écrit avant la généricité doit pouvoir être utilisé avec la version de Java qui offre la généricité La solution choisie pour respecter cette contrainte est appelée « effacement de type » (type erasure)



List<Employe> employes = new ArrayList<Employe>(); Employe e = new Employe("Dupond"); employes.add(e); // On ajoute d'autres employés ... for (int i = 0; i < employes.size(); i++) { System.out.println(employes.get(i).getNom()); }</pre> Richard Grin Généricié page 63



Des complications • Cet effacement implique des compromis et des impossibilités qui compliquent l'utilisation de la généricité en Java