SERVICE REUSABILITY: PARAMETERIZATION

116

116

Parameterized Types

```
public interface Dictionary <K, T> {
  public void put (K k, T v) { … }
  public T get (K k) { … }
}

Dictionary<String, Integer> salary =
     new Hashtable<String, Integer>();
salary.put (“John Doe”, new Integer(70000));
…
Integer jdSalary = salary.get(“John Doe”);
```

117

117

Unsafe Polymorphism

- Java: If T1 ≤ T2, then T1[] ≤ T2[]

```
void printList (Object[] list) { ... }
String[] L = { "Hiyo", "Silver", "Away!" }
printList(L);
```

118

118

Unsafe Polymorphism

- Java: If T1 ≤ T2, then T1[] ≤ T2[]

```
void printList (Object[] list) { ... }
String[] L = { "Hiyo", "Silver", "Away!" }
printList(L);

Integer[] X = { new Integer(3); }
Object[] Y = X;
Object Z = "Hello";
Y[0] = Z;
X[0].intValue() + 7;
```

119

119

Bounded Parametric Polymorphism

```
interface Printable {
  void print (OutputStream os);
}

void printList <A ≤ Printable> (A[] L) {
  ...L[i].print(os) ...
}

class P implements Printable { ... }
P[] X = { new P() };
printList(X);
```

120

120

SERVICE REUSABILITY: SUBTYPING AND INHERITANCE

121

121

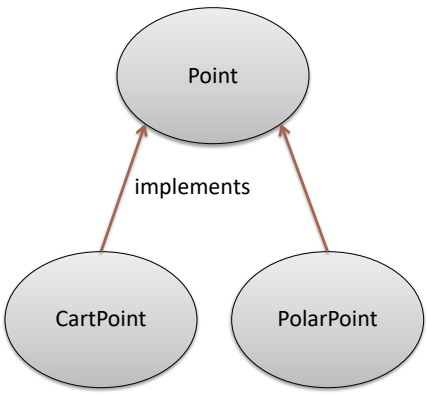
Subtyping and Inheritance

```
public interface Point {
  public float getX();
  public float getY();
}
class CartPoint implements Point {
  private float x, y;
  public float getX() { return x; }
  public float getY() { return y; }
}
class PolarPoint implements Point {
  private float rho, theta;
  public float getX() {
    return rho * Math.cos(theta);
  }
  public float getY() {
    return rho * Math.sin(theta);
  }
}
```

122

122

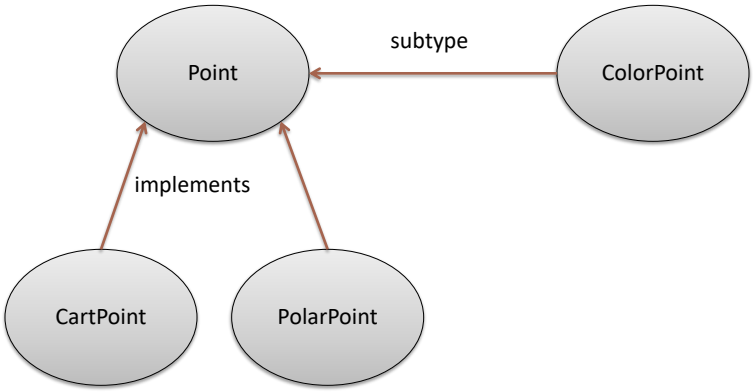
Subtyping and Inheritance



123

123

Subtyping and Inheritance



124

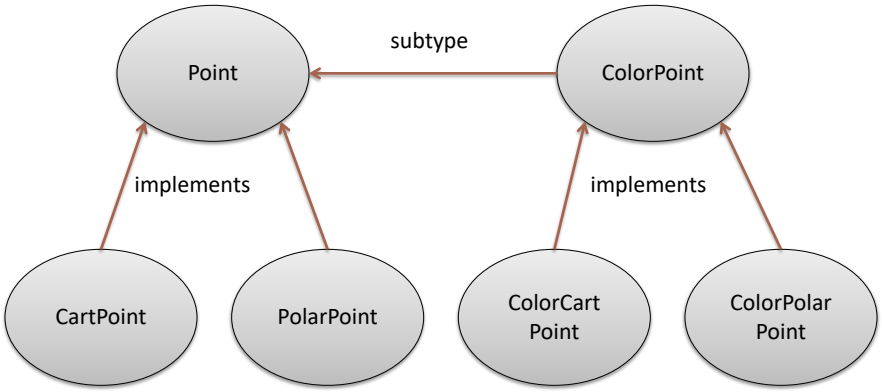
Subtyping and Inheritance

```
public interface Point {
  public float getX();
  public float getY();
}
class CartPoint implements Point {
  private float x, y;
  public float getX() { return x; }
  public float getY() { return y; }
}
class PolarPoint implements Point {
  private float rho, theta;
  public float getX() {
    return rho * Math.cos(theta);
  }
  public float getY() {
    return rho * Math.sin(theta);
  }
}
```

```
public interface ColorPoint
  extends Point {
  public Color getColor();
}
class ColorCartPoint
  extends CartPoint
  implements ColorPoint {
  private Color c;
  public float getColor() {
    return c;
  }
}
class ColorPolarPoint
  extends PolarPoint
  implements ColorPoint {
  private Color c;
  public float getColor() {
    return c;
  }
}
```


125

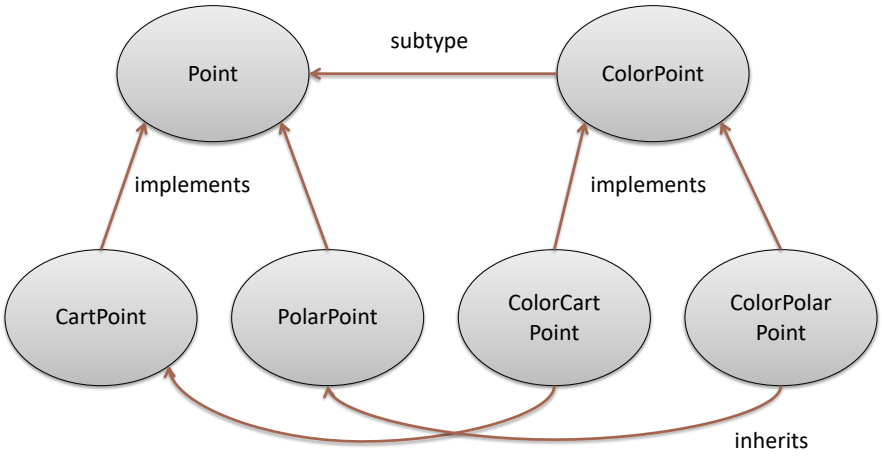
Subtyping and Inheritance



126

126

Subtyping and Inheritance

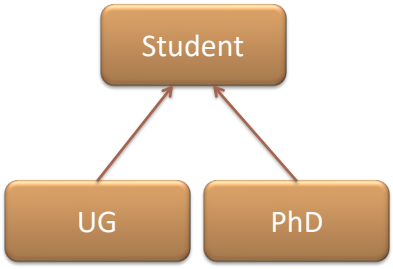


127

127

Uses of Subtyping: Specialization

```
class Student {
  int CWID;
  String name;
  Faculty advisor;
}
class UG extends Student {
  String year;
  String major;
}
class PhD extends Student {
  String thesisTitle;
  List<Faculty> committee;
}
```



128

128

Uses of Subtyping: Mixins

```
interface FacultyBase {
  int getCWID();
  String getName();
}
interface Advisor extends FacultyBase {
  List<Student> getAdvisees();
}
interface Researcher extends FacultyBase {
  List<Grant> getGrants();
}
interface Faculty
extends Advisor, Researcher { }
```

129

129

SERVICE REUSABILITY: SERVICE SUBTYPING

130

130

Subtyping of Service Operation Types

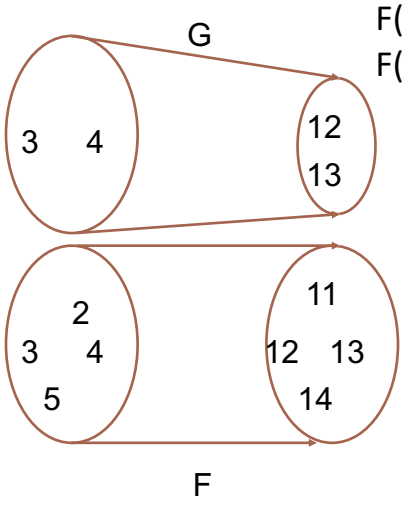
```
// X has type {11,12,13,14}
// Y has type {12,13}
X = Y; // safe
Y = X; // unsafe!

// F has type
//       {2,3,4,5} → {11,12,13,14}
// G has type {3,4} → {12,13}
F = G; // Is it safe?
```

131

131

Subtyping of Service Operation Types

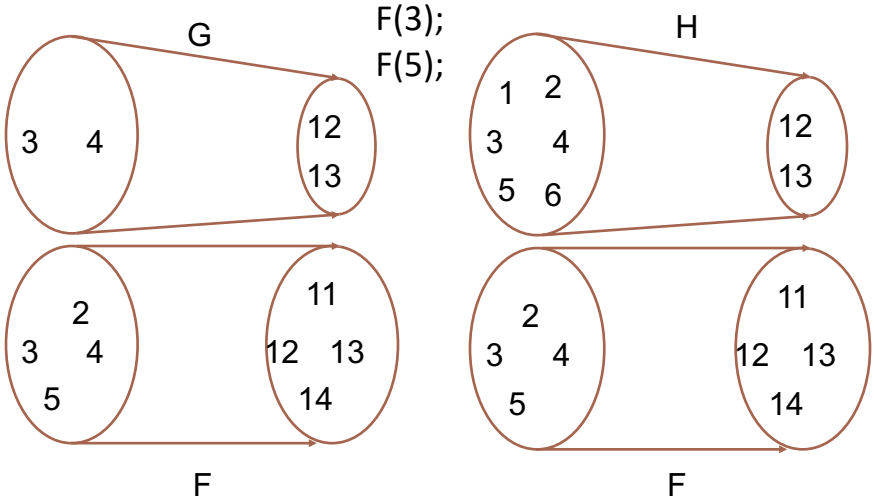
F = G;
F(3);
F(5);



132

Subtyping of Service Operation Types

F = H;
F(3);
F(5);



133

Subtyping Rule

- Subtyping of service operation types:
  - $(T_1 \rightarrow T_2) \leq (T_1' \rightarrow T_2')$
- If
  - Subtyping is **covariant** in result type ($T_2 \leq T_2'$)
  - Subtyping is **contravariant** in argument type ($T_1' \leq T_1$)

134