Recap: References

Classes and Reference Types

- every object belongs to a class
- classes act like types; for instance, references to an object are given a particular type when we declare it:

```
...
    Robot c3po = new Robot("C3PO");
...
```

the reference to ‘type’ Robot

here we specify the type of object the reference c3po can point to

the constructor called must then relate to the type we specified

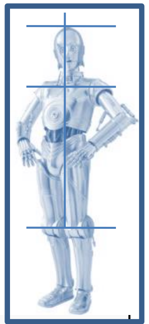
→ What effects can we achieve by introducing sub-class relationships?

```
class Robot {
  String name;
  int numLegs;
  float powerLevel;

 Robot(String productName) {
    name = productName;
    numLegs = 2;
    powerLevel = 2.0f;
  }

  void talk(String phrase) {
    if (powerLevel >= 1.0f) {
      System.out.println(name + " says " + phrase);
      powerLevel -= 1.0f;
    } else {
      System.out.println(name +
      " is too weak to talk.");
    }
  }

  void charge(float amount) {
    System.out.println(name + " charges.");
    powerLevel += amount;
  }
}
```



Sub-classes

Is one robot class enough?

- According to **Wookieepedia**, 3PO-series droids are “fluent in over six million forms of communication”, weigh around 77.6 kg, and have a maximal speed of 21km/hr

- So … they have a lot of specific functionality and might be considered a special class of robot …

Fighting Code Duplication

- Problem: you have written a class (e.g. **Robot**), which almost does what you want, but requires some extensions
- Idea: **extend** features from the existing class by creating a **child class** that automatically receives all features of the parent class (e.g. **name**, **talk()**,...) without writing code again
- Implementation: you define a new class (e.g. **TranslationRobot**) inheriting **all** features from the existing parent class, but add or adapt features so that the new class does exactly what you want
- Result: leads to **DRY** (do-not-repeat-yourself) code where each feature has a **single code source**

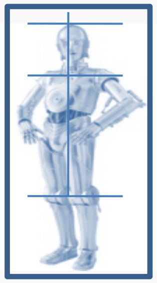
parent class Robot provides all its features to the child class

```
class Robot {
  String name;
  int numLegs;
  float powerLevel;

 Robot(String productName) {
    name = productName;
    numLegs = 2;
    powerLevel = 2.0f;
  }

  void talk(String phrase) {
    if (powerLevel >= 1.0f) {
      System.out.println(name+" says "+
                          phrase);
      powerLevel -= 1.0f;
    } else {
      System.out.println(name +
      " is too weak to talk.");
  } }

  void charge(float amount) {
    System.out.println(name+" charges.");
    powerLevel += amount;
} }
```



'extends' signals inheritance from Robot class

```
public class TranslationRobot extends Robot {
  // class has everything that Robot has implicitly
  String substitute; //and more features

  TranslationRobot(String substitute) {
    this.substitute = substitute;
  }

  void translate(String phrase) {
    this.talk(phrase.replaceAll("a", substitute));
  }

  @Override
  void charge(float amount) { //overriding
    System.out.println(name + " charges double.");
    powerLevel = powerLevel + 2 * amount;
} }
```

added method here

parent method is replaced here

Usage of Inheritance

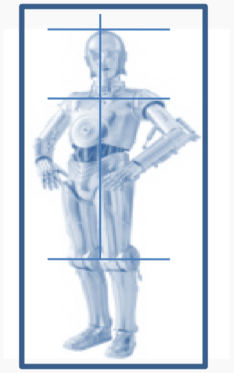
- a child class can provide new or alter old functionality by…
  (1) adding extra attributes
  (2) adding extra methods
  (3) replacing existing methods
  (known as overriding)
- inheritance renders classes more re-usable by making them ‘extendable’ and ‘adaptable’
- WARNING: Only use inheritance for specialisation, i.e. when there is an **is-a relationship**, not a **has-a relationship** (e.g. a Motor class should not be parent to a Robot class)

```java
class Robot {
  String name;
  int numLegs;
  float powerLevel;

 Robot(String productName) {
    name = productName;
    numLegs = 2;
    powerLevel = 2.0f;
  }

  void talk(String phrase) {
    if (powerLevel >= 1.0f) {
      System.out.println(name+" says "+
                           phrase);
      powerLevel -= 1.0f;
    } else {
      System.out.println(name +
      " is too weak to talk.");
  } }

  void charge(float amount) {
    System.out.println(name+" charges.");
    powerLevel += amount;
} }
```



```java
public class TranslationRobot extends Robot {
  // class has everything that Robot has implicitly
  String substitute; //and more features

  TranslationRobot(String substitute) {
    this.substitute = substitute;
  }

  void translate(String phrase) { //added method
    this.talk(phrase.replaceAll("a", substitute));
  }
  @Override
  void charge(float amount) { //overriding
    System.out.println(name + " charges double.");
    powerLevel = powerLevel + 2 * amount;
} }
```

```java
public class InheritanceWorld {

  public static void main (String[] args) {
    TranslationRobot c3po = new TranslationRobot("e");
    c3po.translate("'This text is translated.'");
} }
```

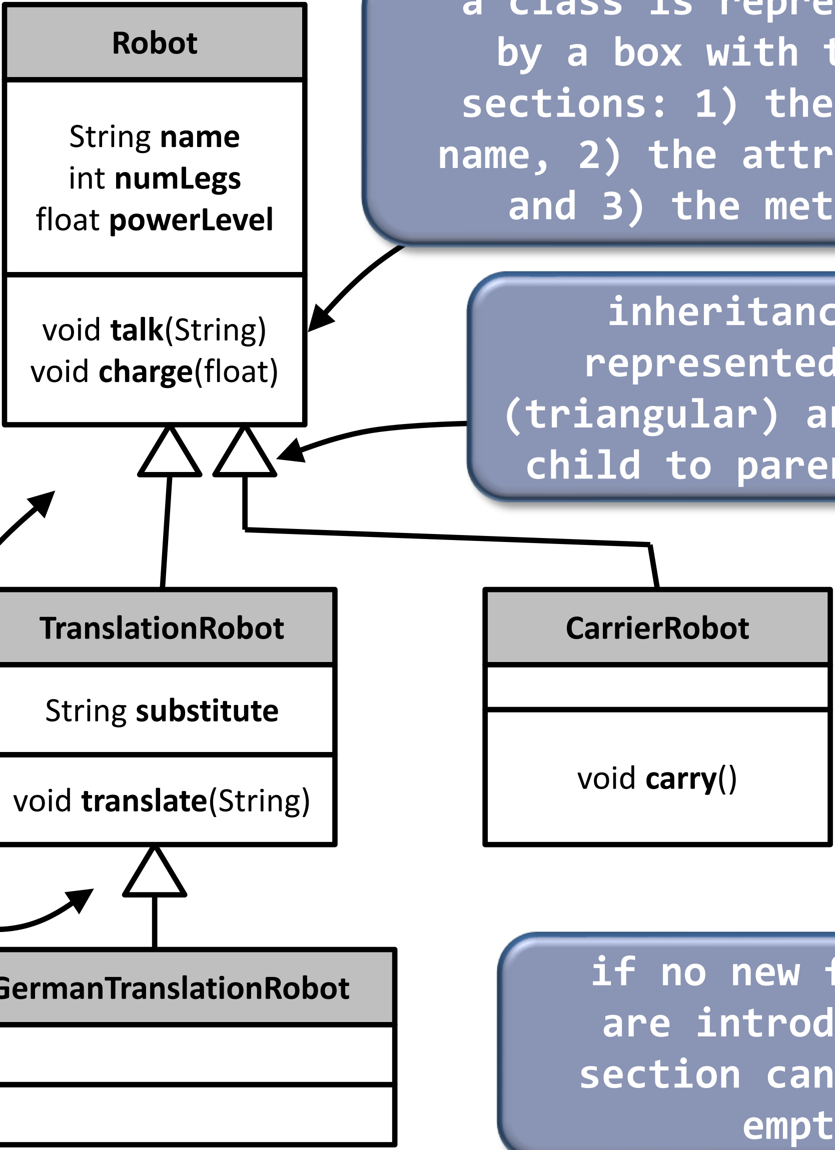
Class Hierarchies

Sketching Inheritance Relationships

This notation borrows from the Unified Modelling Language (UML), which is an established standard for modelling systems, particularly object-oriented ones...

a class is represented by a box with three sections: 1) the class name, 2) the attributes, and 3) the methods

inheritance is represented by a (triangular) arrow from child to parent class



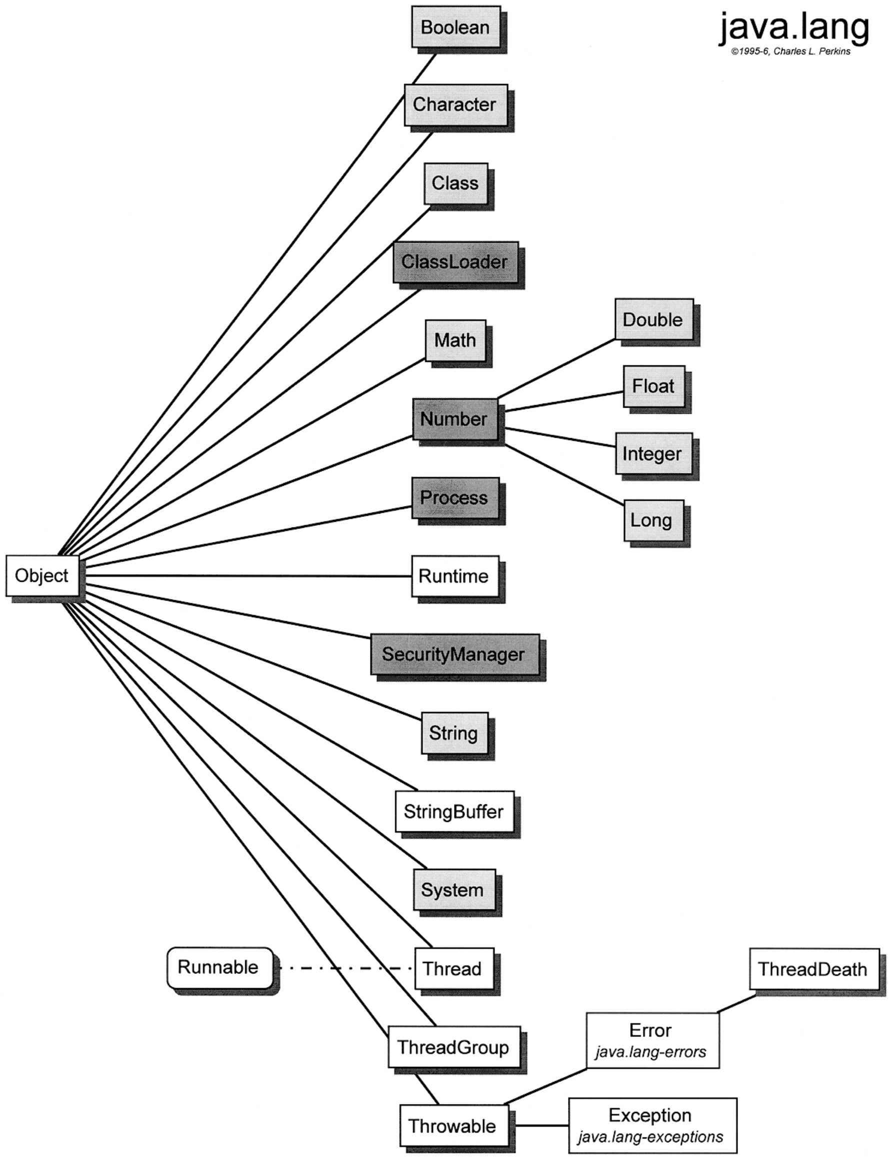
multi-level inheritance

Why could a class with no new features be a valid and useful child class?

if no new features are introduced, a section can be left empty

The Class Hierarchy

- in **single** inheritance, as in Java, a class is derived from one direct super class only
- the resulting **class hierarchy** defines the inheritance relationship between all classes in a **tree structure**
- the root of the class hierarchy is the class **`Object`**
- every class in **Java** directly (implicitly, if it has no parent) or indirectly via multi-level inheritance extends (inherits from) the class **`Object`**


java.lang
©1995-6, Charles L. Perkins
Object
Boolean
Character
Class
ClassLoader
Math
Number
Double
Float
Integer
Long
Process
Runtime
SecurityManager
String
StringBuffer
System
Runnable
Thread
ThreadGroup
Throwable
Error
java.lang-errors
ThreadDeath
Exception
java.lang-exceptions