Review

- Inheritance:
  - Represents the "is-a" relationship between classes
  - Represents specialization of classes (subsets)
  - Represents a way of describing alternatives (alternative subclasses)
  - Is a mechanism for reusability
- Syntax:

```
class B { ... }
class A extends B { ... }
```

- A is a *subclass* of B, or equivalently, A is *derived from* B, A is a *child of* B, or B is a *superclass of* A, or B is a *parent of* A.
- Means that the set of A objects is a subset of the set of B objects.

```
class Labrador extends Dog { ... }
```

Inheritance

- Classes as sets of objects:
  - "is-a" between an object and a class is the same as $\in$
  - "is-a" between two classes is the same as $\subseteq$

- Let A, B, C be sets
  - If $A \subseteq B$ and $x \in A$ then $x \in B$
  - If $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$
  - If $B \subseteq A$ and $C \subseteq A$, and there is no other set D such that $D \subseteq A$ then $A = B \cup C$

Inheritance

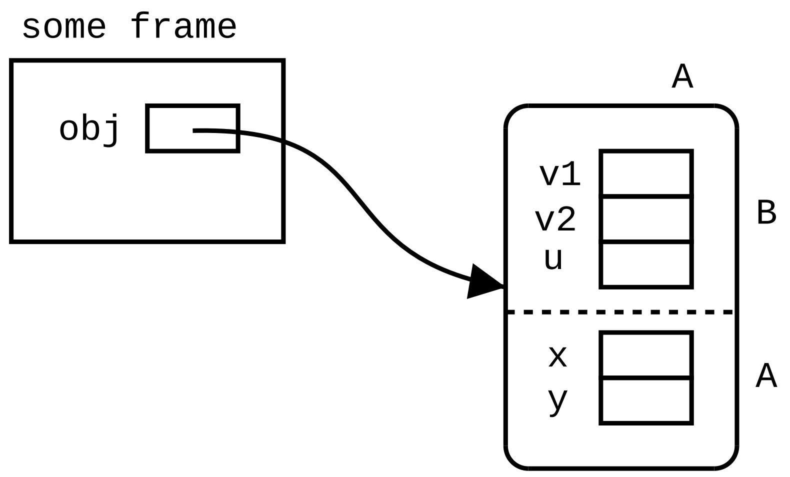
- Inheritance (is-a) and aggregation (has-a):
  - If A is a subclass of B then A has all the attributes and method of B, and it may have more.
  - If every A *is a* B and every B *has a* C then every A *has a* C
  - If every labrador is a dog and every dog has a tail then every labrador has a tail.

Inheritance

```
class C { ... }
class D { ... }
class E { ... }
class B {
  C v1, v2;
  D u;
  void m() { ... }
}
class A extends B {
  E x;
  C y;
  void p() { ... }
  void s() { ... }
}
```

Inheritance

```
// In some client
A obj = new A();
obj.p();
obj.m();
// We can refer to ... obj.x ... obj.y ...
// ... obj.u ... obj.v1 ... obj.v2 ...
```



Accessing a method or attribute

- Method (and attribute) lookup:
  - If a method (or attribute) m is applied to an object of type A the method m of class A is executed (or accessed) if it is declared in A.
  - If m is not defined in A and A is a subclass of B then the method m of class B is executed if it is declared in B.
  - If m is not defined in B and B is a subclass of C then the method m of class C is executed if it is declared in C.
  - ...
  - If no "ancestor" of A has a definition of method m then an error occurs.

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
}
class B extends A {
    void p()
    {
        System.out.println("2 ");
        m();
    }
}
```

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        A obj1 = new A();
        B obj2 = new B();
        obj1.m();
        obj2.m();
        obj2.p();
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    int x = 3;
    void m()
    {
        System.out.println(x);
    }
}
class B extends A {
    void p()
    {
        System.out.println(x);
    }
}
```

Inheritance

- *Shadowing a variable*: if class A has an attribute n and a subclass B of A also declares an attribute n, then n of B *shadows* n of A.

```
class A {
    int x = 3;
}
class B extends A {
    int x = 5;
}
```

- If an instance of B is created it will contain both variables. Shadowed variables are also inherited, but can be accessed only by using the special reference `super`.

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    int x = 3;
    void m()
    {
        System.out.println(x);
    }
}
class B extends A {
    int x = 5;
    void p()
    {
        System.out.println(x);
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    int x = 3;
    void m()
    {
        System.out.println(x);
    }
}
class B extends A {
    int x = 5;
    void p()
    {
        System.out.println(super.x);
    }
}
```

Inheritance

- *Overriding a method*: if class A has a method m and a subclass B of A also declares a method called m, then m of B *overrides* m of A.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
}
class B extends A {
    void m()
    {
        System.out.println("2 ");
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
}
class B extends A {
    void m()
    {
        System.out.println("3 ");
    }
    void p()
    {
        System.out.println("2 ");
        m();
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
}
class B extends A {
    void m()
    {
        System.out.println("3 ");
    }
    void p()
    {
        System.out.println("2 ");
        super.m();
    }
}
```

Inheritance

- A method in a superclass can access *indirectly* the attributes and methods of a subclass (but only those which have been overriden.)

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
    void p()
    {
        System.out.print("2 ");
        m();
    }
}
class B extends A {
    void m()
    {
        System.out.println("3 ");
    }
}
```

Inheritance

- A method in a superclass can access *indirectly* the attributes and methods of a subclass.

```
public class Inh1
{
    public static void main(String[] args)
    {
        A obj1 = new A();
        B obj2 = new B();
        obj1.m();
        obj2.m();
        obj1.p();
        obj2.p();
    }
}
```

Polymorphism

- Polymorphism means “many forms.”
- Polymorphism is the characteristic of being able to assign a different meaning or usage to something in different contexts
- If a class A has a method m we could give different meaning to m by defining subclasses that override m, and therefore the result of executing m depends on the context, since the context decides which subclass is instantiated.

Polymorphism

```
class Creature {
  boolean alive;
  void move()
  {
    System.out.println("The way I move is by...”);
  }
}
class Human extends Creature {
  void move()
  {
    System.out.println("Walking...");
  }
}
class Martian extends Creature {
  void move()
  {
    System.out.println("Crawling...");
  }
}
```

Polymorphism

```
public class ZooTest {
  public static void main(String[] args)
  {
    Human yannick = new Human();
    Martian ernesto = new Martian();
    ernesto.move();
    yannick.move();
  }
}
```

Polymorphism

- A polymorphic method is a method which can accept more than one type of argument
- Kinds of polymorphism:
  - Overloading (Ad-hoc polymorphism): redefining a method in the same class, but with different signature (multiple methods with the same name.) Different code is required to handle each type of input parameter.
  - Parametric polymorphism: a method is defined once, but when invoked, it can receive as arguments objects from any subclass of its parameters. The same code can handle different types of input parameters.

Polymorphism

```
class Creature {
  boolean alive;
  void move()
  {
    System.out.println("The way I move is by...”);
  }
}
class Human extends Creature {
  void move()
  {
    System.out.println("Walking...");
  }
}
class Martian extends Creature {
  void move()
  {
    System.out.println("Crawling...");
  }
}
```

Ad-hoc Polymorphism (Overloading)

```
class Zoo {
  void animate(Human h)
  {
    h.move();
  }
  void animate(Martian m)
  {
    m.move();
  }
}

public class ZooTest {
  public static void main(String[] args)
  {
    Zoo my_zoo = new Zoo();
    Human yannick = new Human();
    Martian ernesto = new Martian();
    my_zoo.animate(ernesto); // Polymorphic call
    my_zoo.animate(yannick);  // Polymorphic call
  }
}
```

Ad-hoc Polymorphism (Overloading)

```
class Penguin extends Creature {
  void stumble()
  {
    System.out.println(“Ouch”);
  }
}

class Zoo {
  void animate(Human h)
  {
    h.move();
  }
  void animate(Martian m)
  {
    m.move();
  }
  void animate(Penguin p)
  {
    p.move();
  }
}
```

Parametric Polymorphism

```
class Zoo {
  void animate(Creature c)
  {
    c.move();
  }
}

public class ZooTest {
  public static void main(String[] args)
  {
    Zoo my_zoo = new Zoo();
    Human yannick = new Human();
    Martian ernesto = new Martian();
    my_zoo.animate(ernesto); // Polymorphic call
    my_zoo.animate(yannick);  // Polymorphic call
  }
}
```

Parametric Polymorphism

```
class Zoo {
  void animate(Creature c)
  {
    c.move(); // Dynamic-dispatch
    // move *must* be defined in class Creature
  }
}

public class ZooTest {
  public static void main(String[] args)
  {
    Zoo my_zoo = new Zoo();
    Human yannick = new Human();
    Martian ernesto = new Martian();
    Penguin paco = new Penguin();

    my_zoo.animate(ernesto);
    my_zoo.animate(yannick);
    my_zoo.animate(paco);
  }
}
```

Accessing super

```
class Human extends Creature {
  void move()
  {
    super.move();
    System.out.println(“Walking...”);
  }
}
class Martian extends Creature {
  void move()
  {
    super.move()
    System.out.println(“Crawling...”);
  }
}
```

Polymorphism

- Polymorphism is a tool that permits abstraction and reusability
- A polymorphic method is a method which can receive as input any object whose class is a subclass of the methods's parameter.
- Ad-hoc polymorphism is overloading (providing separate methods for each expected parameter type)
- Parametric polymorphism relies on dynamic-dispatching. Dynamic-dispatching is the process by which the runtime system directs the message of an object to the appropriate subclass.
- A dynamic-dispatch can be decided only at run-time, not at compile-time, because the compiler cannot know which is the actual object passed as argument to a polymorphic method. Furthermore, the same method might be called with different objects from different classes during the execution of the program.

The end