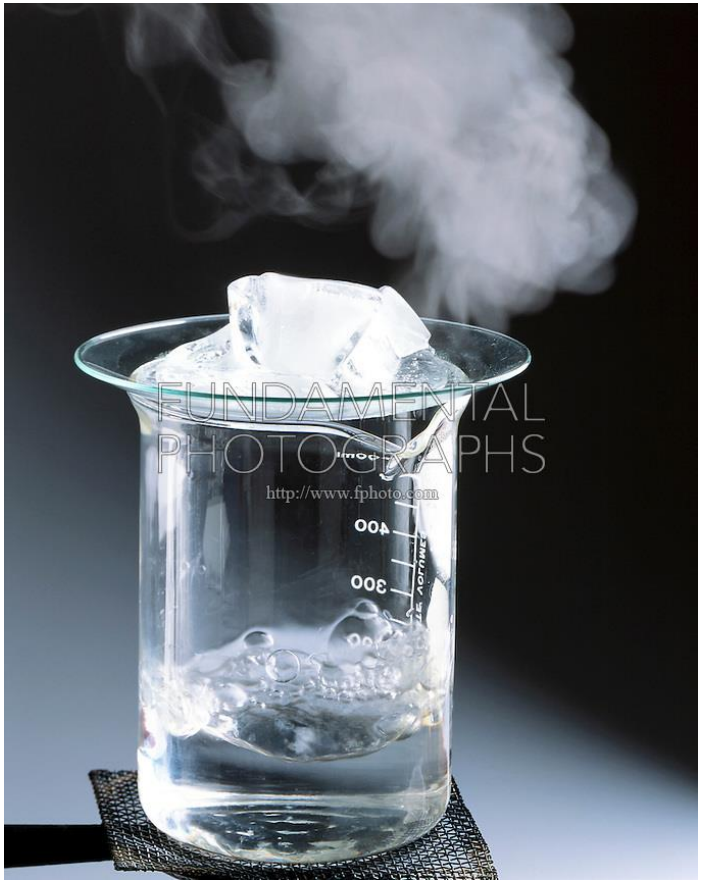
Object Oriented Programming

Polymorphism : Casting

Polymorphism



- In chemistry they talk about polymorph materials as an example H_20 is polymorph (ice, water, and steam).

- *Polymorphism:* "The ability of a variable or argument to refer at run-time to instances of various classes" [Meyer pp. 224].

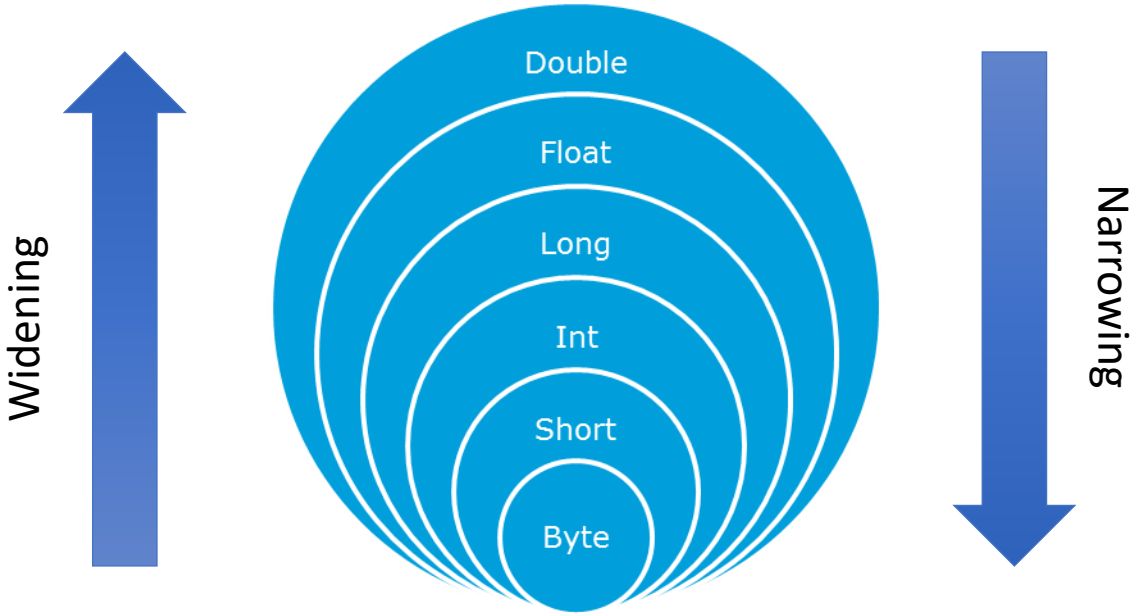
Casting

- taking an Object of one particular type and "turning it into" another Object type
- Type Casting
- Object Casting

Type Casting

- Assigning a value of one type to a variable of another type
- Widening Casting
  - Implicit
  - Automatic type conversion
  - The two types must compatible
  - The target type is larger than the source type
- Narrowing Casting
  - Explicit

Type Casting



Widening Casting

```
public class Driver{
  public static void main(String args[]){
    int i = 50;
    long l = i;
    float f = l;

    System.out.println("int value: " + i);
    System.out.println("long value: " + l);
    System.out.println("float value: " + f);
  }
}
```

```
> int value: 50
> long value: 50
> float value: 50
```

byte → short → int → long → float → double

Widening

Narrowing Casting

```
public class Driver{
  public static void main(String args[]){
    double d = 25.16;
    long l = (long)d;
    int i = (int)l;

    System.out.println("double value: " + d);
    System.out.println("long value: " + l);
    System.out.println("int value: " + i);
  }
}
```

```
> double value: 25.16
> long value: 25
> int value: 25
```

byte ← short ← int ← long ← float ← double

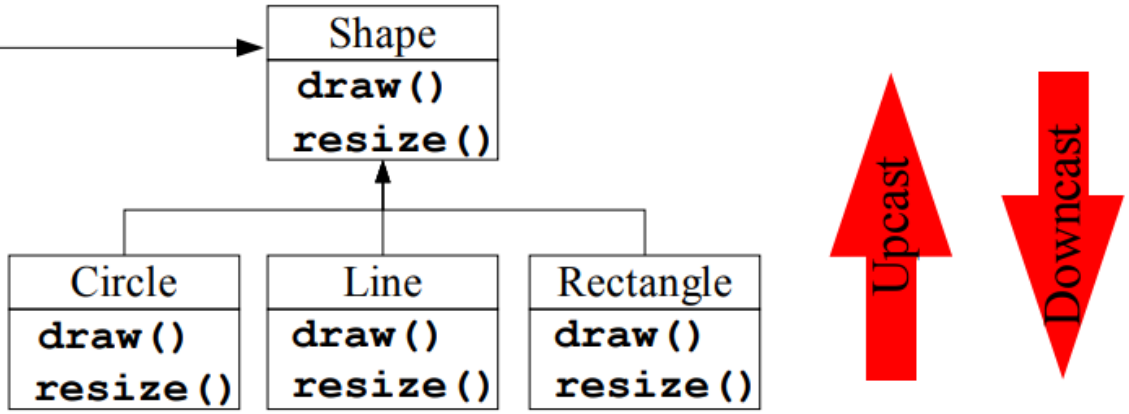
Narrowing

Object Casting

- Turn an object of a class to another class
- Both Classes should have inheritance or implement relationship
- Upcasting
  - Child object casted to parent class reference variable
- Downcasting
  - Object casted back to child class reference variable
- Virtual Method Invocation

Upcast and DownCast

```
// common interface
Shape s;
s.draw()
s.resize()
```




```
// upcasting
Shape s = new Line();
s.draw()
s.resize()

Shape s = new Line();
Line l = (Line) s; // downcast
```

Upcast and DownCast

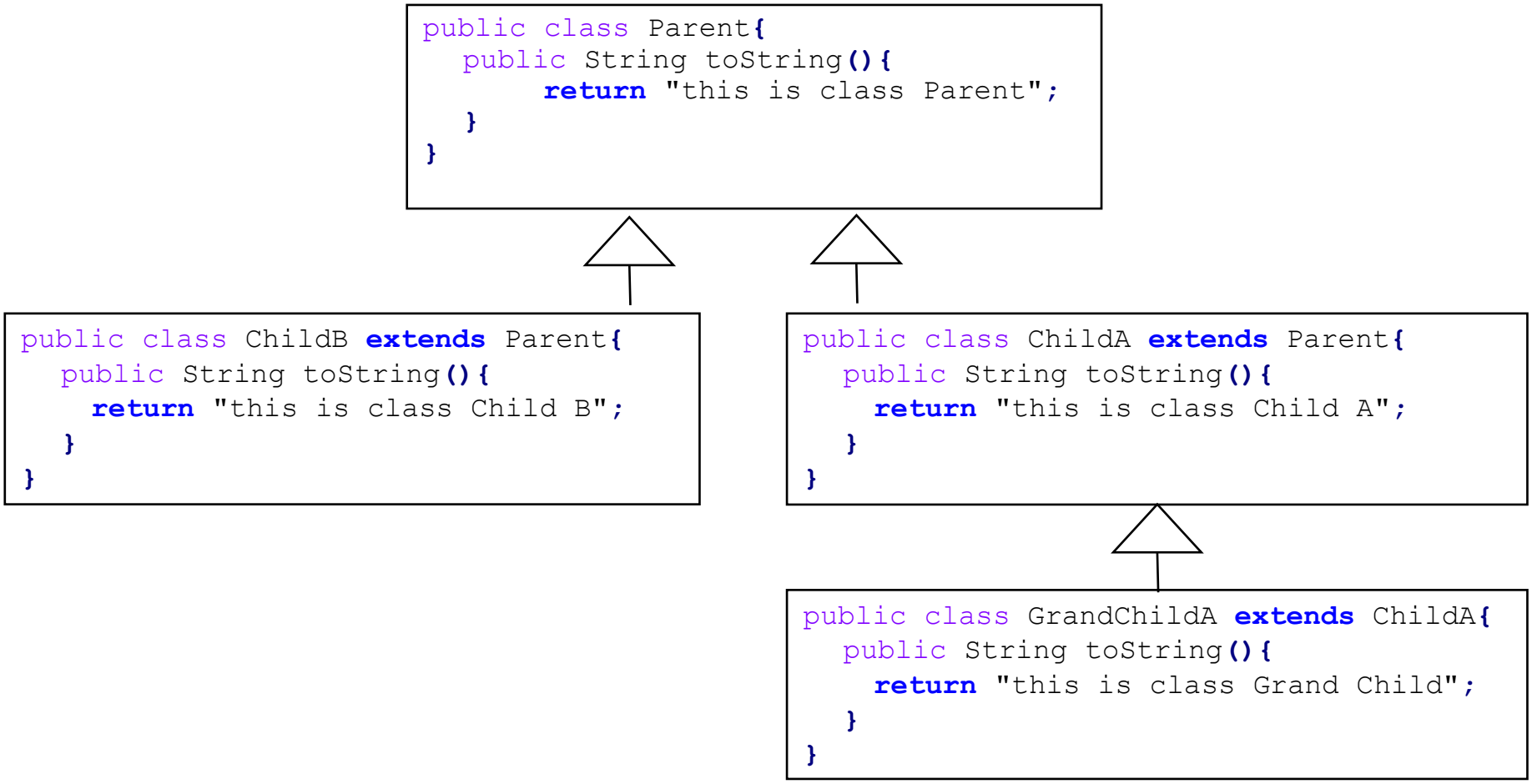
```
Shape s = new Shape();
Circle c = new Circle();
Line l = new Line();
Rectangle r = new Rectangle();

s = l;          // is this legal?
l = s;          // is this legal?
l  = (Line)s    // is this legal?
```

- The assignment **`s = l`** is legal if the static type of **`l`** is **`Shape`** or a subclass of **`Shape`**.
- This is *static type checking* where the type comparison rules can be done at compile-time.
- Polymorphism is constrained by the inheritance hierarchy.

Virtual Method Invocation

- Virtual method invocation is the form of **Upcasting**
- At the time of the object that has been created calling overridden method in the parent class, the Java compiler will do the invocation (call) to the overriding method in a subclass, which is supposed to be called is overridden

Example


public class Parent{
public String toString(){
return "this is class Parent";
}
}
public class ChildB extends Parent{
public String toString(){
return "this is class Child B";
}
}
public class ChildA extends Parent{
public String toString(){
return "this is class Child A";
}
}
public class GrandChildA extends ChildA{
public String toString(){
return "this is class Grand Child";
}
}

Example

```
public class Driver{
  public static void main(String args[]){
    Parent p = new Parent();
    ChildA cA = new ChildA();
    ChildB cB = new ChildB();
    GrandChildA gC = new GrandChildA();

    System.out.println(p.toString());

    System.out.println(cA.toString());

    System.out.println(cB.toString());

    System.out.println(gC.toString());
  }
}
```

```
> this is method Parent

> this is method Child A

> this is method Child B

> this is method Grand Child
```

Example VMI/Upcasting

```
public class Driver{
  public static void main(String args[]){
    Parent castP;

    castP = new Parent();
    System.out.println(castP.toString());

    castP = new ChildA();
    System.out.println(castP.toString());

    castP = new ChildB();
    System.out.println(castP.toString());

    castP = new GrandChildA();
    System.out.println(castP.toString());

  }
}
```

```
> this is method Parent

> this is method Child A

> this is method Child B

> this is method Grand Child
```

Example VMI /Upcasting

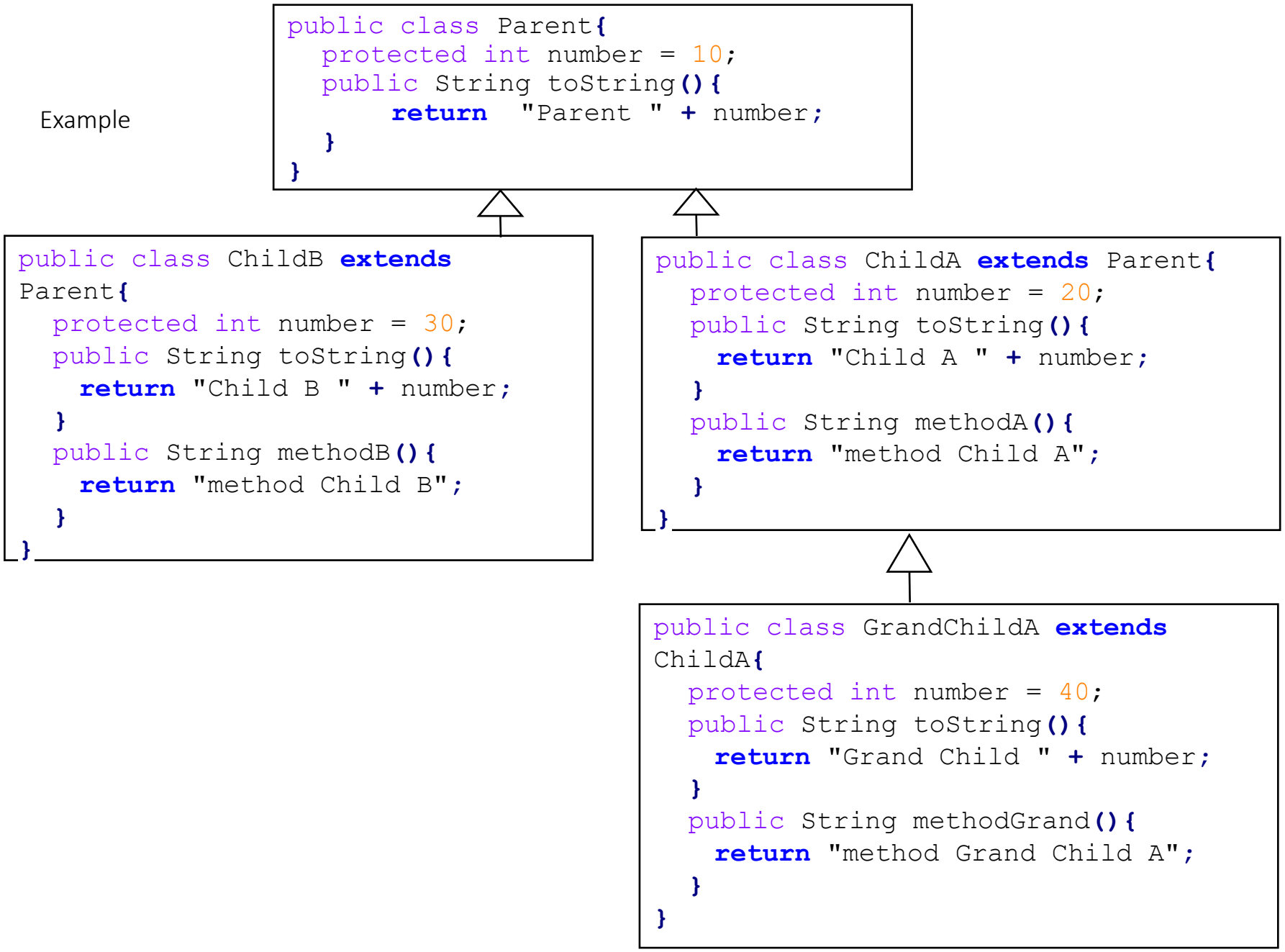
```
public class Driver{
  public static void main(String args[]){
    Parent castP;
    GrandChildA gC = new GrandChildA();
    ChildB cB = new ChildB();

    castP = gC;
    System.out.println(castP.toString());

    castP = cB;
    System.out.println(castP.toString());

  }
}
```

```
> this is method Grand Child

> this is method Child B
```

Upcasting

- `castP = new ChildA();`
- Object `castP` has a behavior that is in accordance with the runtime type, not the compile type
- When compile-time `castP` is a Parent
- When runtime `castP` is ChildA
- Therefore
  - `castP` can only access variable Parent
  - `castP` can only access method ChildA

Example


public class Parent{
protected int number = 10;
public String toString(){
return "Parent " + number;
}
}
public class ChildB extends
Parent{
protected int number = 30;
public String toString(){
return "Child B " + number;
}
public String methodB(){
return "method Child B";
}
}
public class ChildA extends Parent{
protected int number = 20;
public String toString(){
return "Child A " + number;
}
public String methodA(){
return "method Child A";
}
}
public class GrandChildA extends
ChildA{
protected int number = 40;
public String toString(){
return "Grand Child " + number;
}
public String methodGrand(){
return "method Grand Child A";
}
}

Example

```
public class Driver{
  public static void main(String args[]){
    Parent p = new Parent();
    ChildA cA = new ChildA();
    ChildB cB = new ChildB();
    GrandChildA gC = new GrandChildA();
    Parent castP;

    System.out.println(p.toString());
    System.out.println(cA.toString());
    System.out.println(cB.toString());
    System.out.println(gC.toString());
  }
}
```

```
> Parent 10
> Child A 20
> Child B 30
> Grand Child 40
```

Example

```
public class Driver{
  public static void main(String args[]){
    Parent p = new Parent();
    ChildA cA = new ChildA();
    ChildB cB = new ChildB();
    GrandChildA gC = new GrandChildA();
    Parent castP;

    castP = cA;
    System.out.println(castP.toString());
    System.out.println(castP.number);

    castP = gC;
    System.out.println(castP.toString());
    System.out.println(castP.number);
  }
}
```

```
> Child A 20
> 10
> Grand Child 40
> 10
```

Example

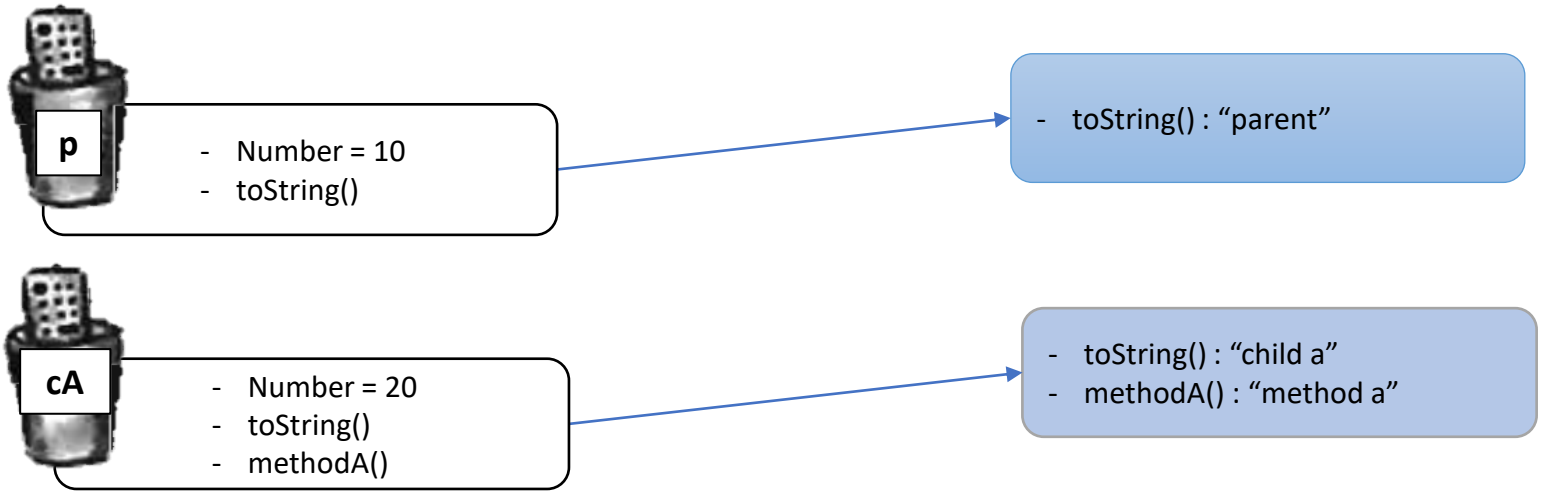
```
public class Driver{
  public static void main(String args[]){
    Parent p = new Parent();
    ChildA cA = new ChildA();
    ChildB cB = new ChildB();
    GrandChildA gC = new GrandChildA();

    System.out.println(cA.methodA());

    Parent castP;
    castP = cA;
    System.out.println(castP.methodA());
  }
}
```

```
> method Child A
```

```
//compile error:
cannot find symbol
```

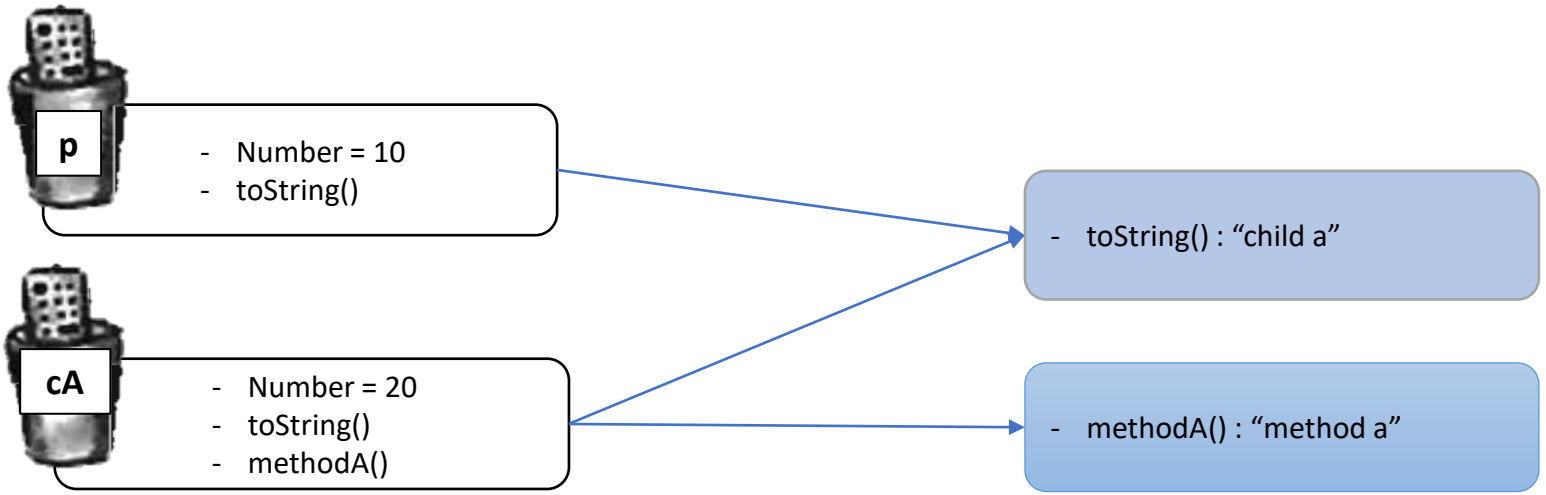
Example - Illustration

```
Parent p;
p = new Parent();

ChildA cA;
cA = new ChildA();
```



Example - Illustration

```
p =  cA ;
System.out.println( p.toString() );
System.out.println( p.number );
```



DownCasting

Returns the upcasted object back to its original class object

Explicit casting

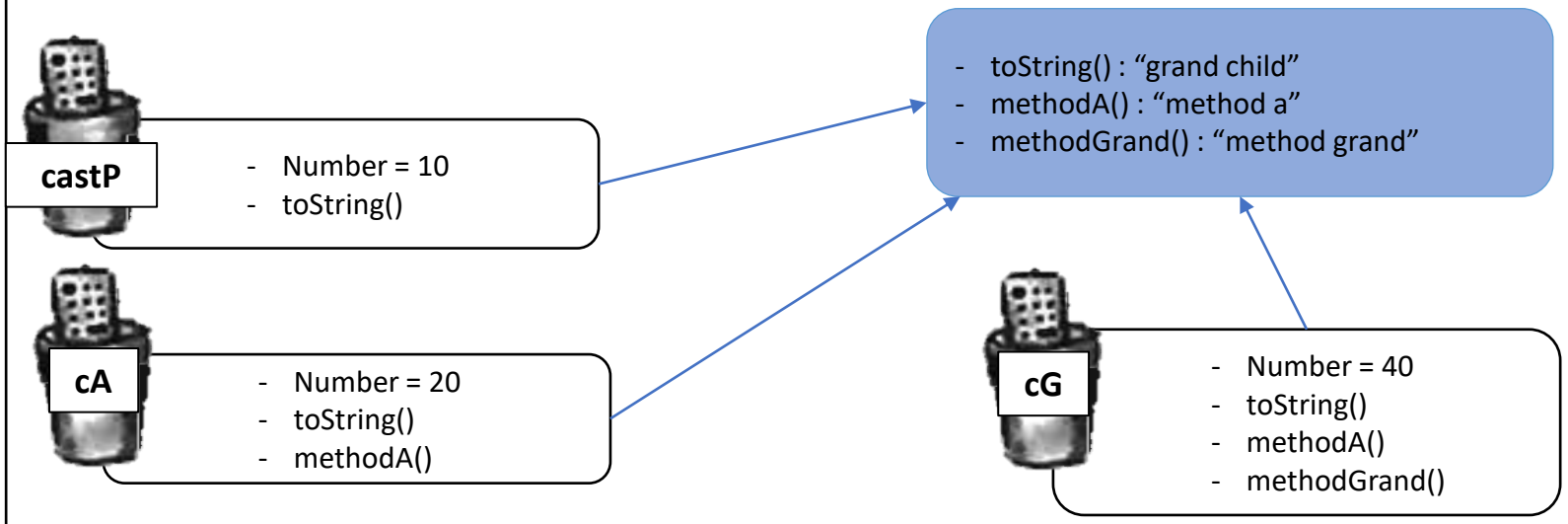
Example

```
public class Driver{
  public static void main(String args[]){
    Parent castP;

    castP = new GrandChildA();
    System.out.println(castP.toString());
    System.out.println(castP.methodA());

    GrandChildA castG = (GrandChildA) castP;
    System.out.println(castG.toString());
    System.out.println(castG.methodA());
    System.out.println(castG.methodGrand());

    ChildA castA = (ChildA) castP;
    System.out.println(castA.toString());
    System.out.println(castA.methodA());

  }
}
```

```
> Grand Child 40
> Compile error

> Grand Child 40
> Method Child A
> Method Grand Child A

> Grand Child 40
> Method Child A
```

Example - Illustration

```
Parent castP;
castP = new GrandChildA();

ChildA cA = (ChildA) castP;
GrandChildA cG = (GrandChildA) castP
```



Example

```
public class Driver{
  public static void main(String args[]){
    Parent castP;
    castP = new GrandChildA();

    System.out.println(castP.toString());

    System.out.println(
          ((ChildA)castP).methodA()
    );

    System.out.println(
          ((GrandChildA)castP).methodGrand()
    );

  }
}
```

```
> Grand Child 40

> Method Child A

> Method Grand Child A
```

Example

```
public class Driver{
  public static void main(String args[]){
    Parent castP;
    ChildA castA;
    GrandChildA gC;

    castP = new ChildB();
    System.out.println(castP.toString());

    castA = (ChildA)castP;
    System.out.println(castA.toString());
    System.out.println(castA.methodA());

  }
}
```

```
> Child B 30

> ClassCastException
```

//runtime error:
Class Cast Exception
Downcast only to
it's original object

Example

```
public class Driver{
  public static void main(String args[]){
    Parent castP;
    ChildA castA;
    GrandChildA gC;

    castP = new ChildA();
    System.out.println(castP.toString());

    gC = (GrandChildA)castP;
    System.out.println(gC.toString());
    System.out.println(gC.methodA());
    System.out.println(gC.methodGrand());
  }
}
```

```
> Child A 20

> ClassCastException
```

//runtime error:
Class Cast Exception
Downcast only to
it's original object

Downcasting

- Can only downcast to original class or its parents' class
- To check whether the object can be casted
  - Use instanceof keyword

Keyword instanceof

- Used to test if an object is of a specified type
  - test if an object is an instance of a class,
  - an instance of a subclass,
  - or an instance of a class that implements a particular interface
- (Object instanceof Class) → boolean

Example

```
System.out.println("p instanceof Parent: "
          + (p instanceof Parent));
System.out.println("p instanceof ChildA: "
          + (p instanceof ChildA));
System.out.println("cA instanceof Parent: "
          + (cA instanceof Parent));
System.out.println("cA instanceof ChildA: "
          + (cA instanceof ChildA));

System.out.println("cB instanceof ChildA: "
          + (cB instanceof ChildA));
System.out.println("cB instanceof Parent : "
          + (cB instanceof Parent));

System.out.println("cG instanceof ChildA: "
          + (cG instanceof ChildA));
System.out.println("cG instanceof Parent: "
          + (cG instanceof Parent));
```

```
> p instanceof Parent: true

> p instanceof ChildA: false

> cA instanceof Parent: true

> cA instanceof ChildA: true


> //compile error, cannot be
converted
> cB instanceof Parent : true


> cG instanceof ChildA: true

> cG instanceof Parent : true
```

Example

```
public class Driver{
  public static void main(String args[]){
    Parent castP;
    ChildA castA;
    GrandChildA gC;

    castP = new ChildA();
    System.out.println(castP.toString());

    if(castP instanceof GrandChildA) {
     gC = (GrandChildA)castP;
     System.out.println(gC.toString());
     System.out.println(gC.methodA());
     System.out.println(gC.methodGrand());
    }
  }
}
```

```
> Child A 40

>
```

Benefits and Downsides

- Flexibility
  - Model Object
- Heterogeneous Collection
- Polymorphic Arguments
- Run-time exception
  - ClassCastException

Heterogeneous Collection

- Collections of objects with different class types

```
public class Driver{
  public static void main(String args[]){
    Parent listP[] = new Parent[4];
     listP[0] = new ChildB();
     listP[1] = new ChildA();
     listP[2] = new Parent();
     listP[3] = new GrandChildA();
  }
}
```

Example

```
for(int i = 0; i < 4; i++){

    System.out.println(i+""+listP[i].toString());

    if(listP[i] instanceof ChildA) {
        ChildA cA = (ChildA)listP[i];
        System.out.println(cA.methodA());
    }

    if(listP[i] instanceof ChildB) {
        ChildB cB = (ChildB)listP[i];
        System.out.println(cB.methodB());
    }

    if(listP[i] instanceof GrandChildA) {
        GrandChildA gC = (GrandChildA)listP[i];
        System.out.println(gC.methodGrand());
    }
}
```

```
> 0 Child B 30
> method Child B

> 1 Child A 20
> method Child A

➢ 2 Parent 10
➢ //tidak masuk if

➢ 3 Grand Child 40
➢ //masuk ke 2 if
> method Child A
> method Grand Child A
```

Polymorphic Arguments

- Method with parent reference as parameter input

```
public void testMethod(Parent p){
  System.out.println(p.toString());

  if(p instanceof ChildA) {
      System.out.println((ChildA)p.methodA());
  } else if(p instanceof ChildB) {
      System.out.println((ChildB)p.methodB());
  } else if(p instanceof GrandChildA) {
      System.out.println((GrandChildA)p.methodGrand());
  }
}
```