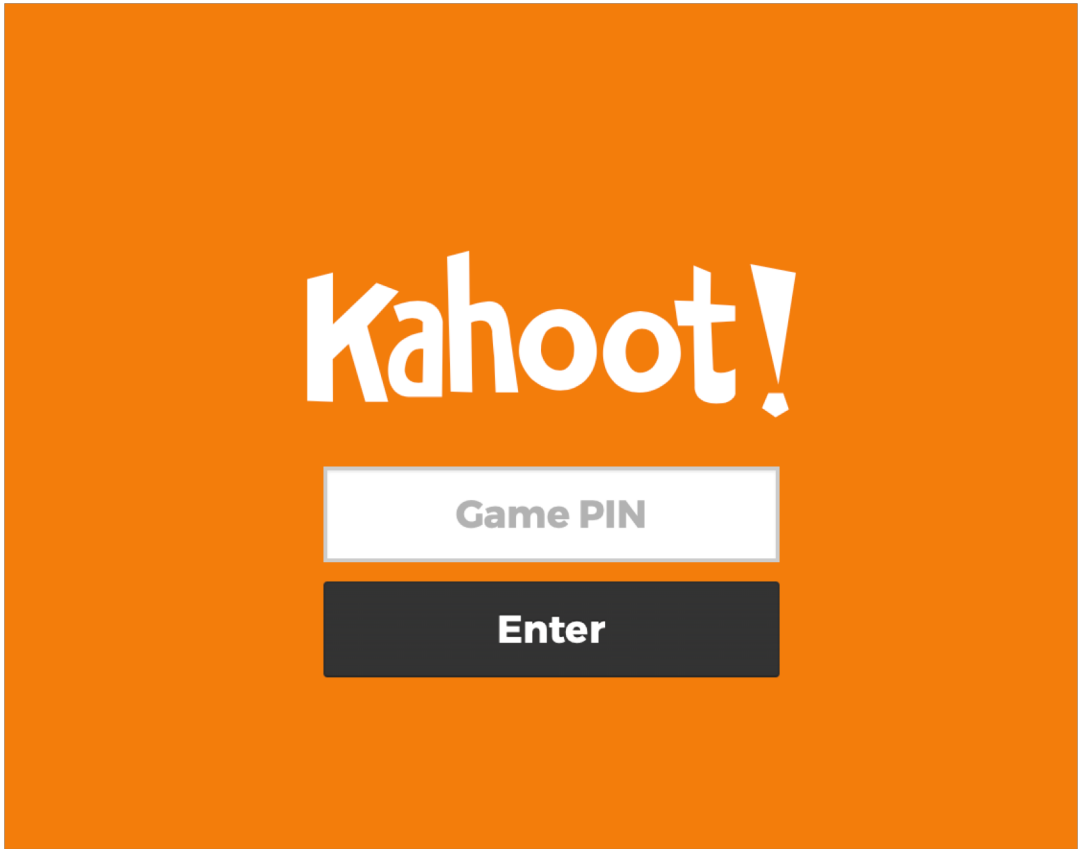
Object orientated programming

polymorphism

Kahoot



Repetition: ArrayList

- The ArrayList class is a resizable array, which can be found in the java.util package.
- The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified
  - If you want to add or remove elements to/from an array, you have to create a new one.
- While elements can be added and removed from an ArrayList whenever you want. The syntax is also slightly different:
- Most important methods: add(), get(), set(), remove()

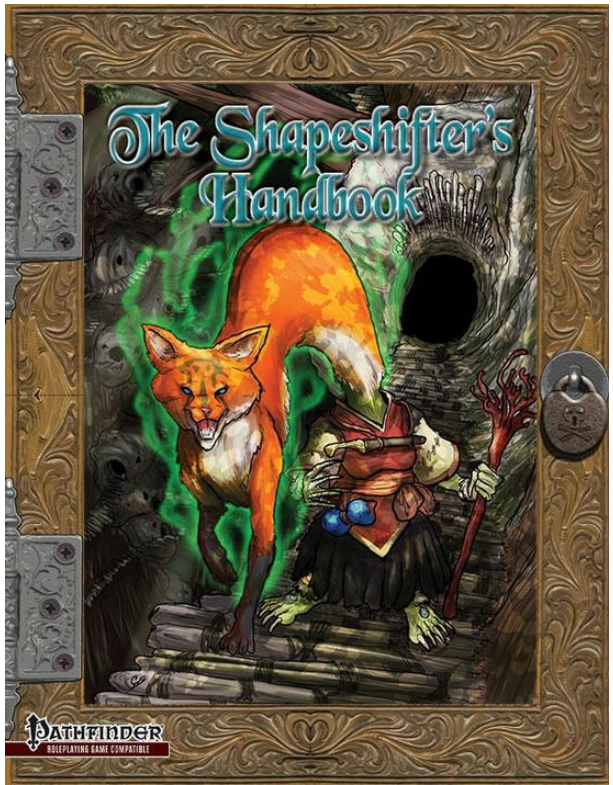
```
import java.util.ArrayList; // import the ArrayList class

ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

- Example: ArrayListTest

Polymorphism

- **Polymorphism** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from two Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.



Polymorphism in Java

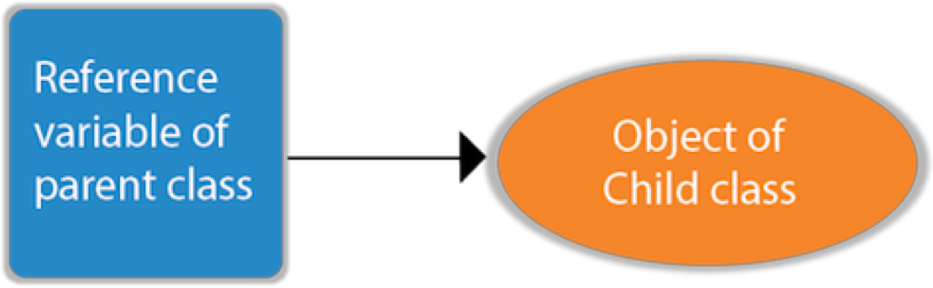
- **In Java there are two types of polymorphism**
- Compile-time polymorphism
  - Resolved at compile time.
  - Overloading a static method is an example of compile time polymorphism.
- Runtime polymorphism
  - Function call will resolved at runtime.
- Polymorphism can be implemented by method overloading and method overriding.

Runtime Polymorphism

- **Runtime polymorphism** (also called **Dynamic Method Dispatch)** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.
- First: understand upcasting before runtime polymorphism

Upcasting

- If the reference variable of a parent class refers to the object of child class, it is known as upcasting.




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

```
A a=new B();//upcasting
```

```
interface I{}
class A{}
class B extends A implements I{}
```

```
B IS-A A
B IS-A I
B IS-A Object
```

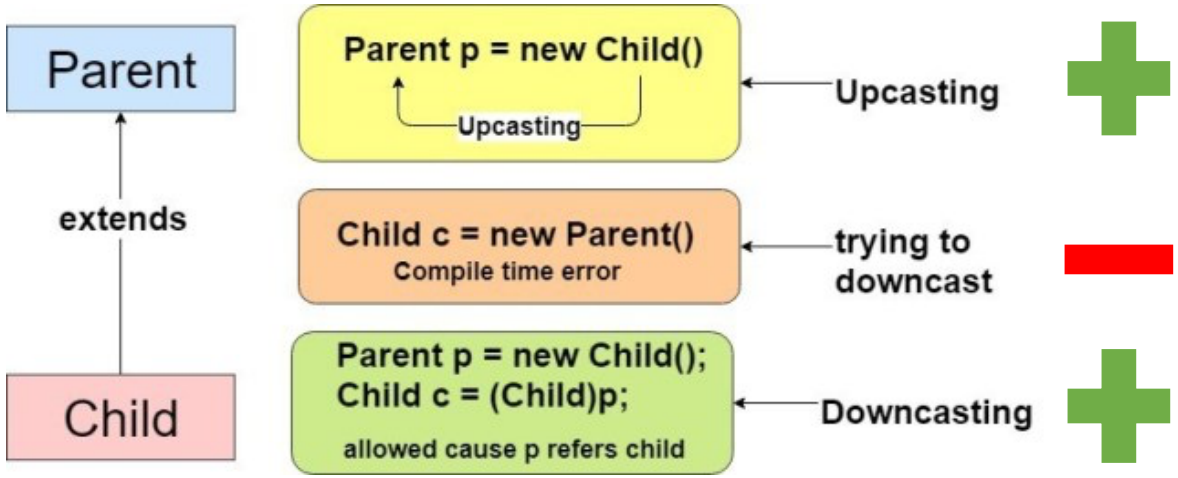
- For upcasting, we can use the reference variable of class type or an interface type.
- Why is B an Object?
  - Since Object is the root class of all classes in Java, so we can write B IS-A Object.

Downcasting

- If the Subclass type refers to the reference, not object, of the parent class, it is known as downcasting.

```
Animal a  = new Dog3();
Dog3 d = (Dog3)a; //Dog 'd' is referring to Animal reference
```

- That means we can downcast only the reference of the Parent class and not an object.



Examples of runtime polymorphism

- An example of two classes Bike and Splendor. Splendor class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of the parent class (com.pgr103.polymorphism.Bike).
  - Since it refers to the subclass object and subclass method overrides the parent class method, the subclass method is invoked at runtime.
  - Since method invocation is determined by the JVM not the compiler, it is known as runtime polymorphism.
- Bank example (again)
  - This example was also presented in method overriding but there was no upcasting used.
  - A Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks. For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, and 9.7% rate of interest (com.pgr103.polymorphism.TestPolymorphism).

More examples of runtime polymorphism

- Shape: A program that allows to “draw” different shapes. The shape class is the parent class implementing a generic draw method. The child classes are implementing the specific drawing methods.
- (com.pgr103.polymorphism.TestPolymorphism2)
- Animal: A program that allows to different types of animals from the parent class Animal. The child classes are implementing the specific eat methods.
- (com.pgr103.polymorphism.TestPolymorphism3)

Java Runtime Polymorphism with Data Members

- A method is overridden, not the data members, so runtime polymorphism cannot be achieved by data members.
- What does that exactly mean?
- Example: Two classes have a data member speedlimit. We are accessing the data member by the reference variable of the parent class which refers to the subclass object. Since we are accessing the data member which is not overridden it will always access the data member of the parent class (com.pgr103.polymorphism.Bike2).
- **Lesson learned: Runtime polymorphism cannot be achieved by data members.**

Java Runtime Polymorphism with Multilevel Inheritance

- Example of Runtime Polymorphism with multilevel inheritance (com.pgr103.polymorphism.Animal2).

- What do we get for this code: com.pgr103.polymorphism.Animal3
- BabyDog is not overriding the eat() method, therefore the eat() method of Dog class is invoked.

Instanceof

- The **instanceof operator** is used to test whether the object is an **instance of the specified type** (class or subclass or interface).
- It is also known as type *comparison operator* because it **compares the instance with type**. It returns either true or false. If the instanceof operator is applied with any variable that has null value, it returns false.
- An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

```
class Simple1{
 public static void main(String args
 Simple1 s=new Simple1();
 System.out.println(s instanceof Sir
 }
}
```

```
class Animal{}
class Dog1 extends Animal{//Dog inherits Animal

 public static void main(String args[]){
 Dog1 d=new Dog1();
 System.out.println(d instanceof Animal);//tru
 }
}
```

```
class Dog2{
 public static void main(String args[]){
  Dog2 d=null;
  System.out.println(d instanceof Dog2);//false
 }
}
```

Downcasting with Java instanceof operator

- When a Subclass type refers to the object of a Parent class, it is known as downcasting.
- If performed directly the compiler gives an error.

```
Dog d=new Animal();//Compilation error
```

- If performed by typecasting, ClassCastException is thrown at runtime.

```
Dog d=(Dog)new Animal();
//Compiles successfully but ClassCastException is thrown at runtime
```

- If instanceof operator is used, downcasting is possible (com.prg103.instanceofexamples.Dog).

Downcasting without the use of java instanceof

- Downcasting can also be performed without the use of instanceof operator (com.prg103.instanceofexamples.Dog1).
- Closer look: The actual object that is referred by a, is an object of the Dog class. If downcasted, it is fine. But what will happen if the following code is run?

```
Animal a=new Animal();
Dog.method(a);
//Now ClassCastException but not in case of instanceof operator
```

Understanding Real use of instanceof in java

- Real use of instanceof keyword by example (com.prg103.instanceofexamples.instanceofTest)
- Second example (com.prg103.instanceofexamples.instanceofTest2)
  - Short explanation of the output: Instances **car** and **moto** are also of type **Vehicle** due to hierarchy, so these assumptions are true. However, **vehicle** is not instance of **Car** (neither **MotorCycle** of course). Also, the instances **vehicle** and **moto** are not instances of **DriveCar**, as only **car** is an instance of that type. Finally, when the **car** gets the null value, it is not an instance of **Vehicle** or **Car** anymore.

Where are we now?

- Constructors
- Static key word
- This key word
- Inheritance
- Aggregation

- Method overloading and overriding in context with OOP
- Polymorphism
- Abstract classes, Interfaces
- Encapsulation, exception handling
- …Object class, cloning, java math…