Why doesn’t it compile?

```java
public class HelloWorld {
    public String message = "Hello World!";
    public HelloWorld(String s) {
        super();
        message = s;
    }
}

public class HelloWorld2 extends HelloWorld {
    public HelloWorld2(String s) {
        super();
        message = s;
    }
}
```
Why doesn’t it compile?

```java
public class HelloWorld {
    public String message="Hello World!";
    private HelloWorld(String s){
        super();
        message=s;
    }
}

public class HelloWorld2 extends HelloWorld{
    public HelloWorld2(String s){
        super(s);
    }
}
```
public class Parent {
    public Parent wave( Parent p) {
        System.out.println("in the parent");
        return null;
    }
}

public class Child extends Parent {
    public Child wave( Parent p) {
        System.out.println("in the child");
        return null;
    }
}

public static void main( String [] args) {
    Parent p = new Child();
    p.wave(p);
}
...and here?

```java
public class Parent {
    public Parent wave(Parent p) {
        System.out.println("in the parent");
        return null;
    }
}

public class Child extends Parent {
    public Child wave(Object o) {
        System.out.println("in the child");
        return null;
    }
    public static void main(String []args) {
        Parent p = new Child();
        p.wave(p);
    }
}
```
What is invalid?

```java
public class HelloWorld{
    static String s="Hello World";
    public static void main(String []args){
        System.out.println(s);
    }
}
```
Expressions

- Method Invocation
- instanceof
- Casts
- blocks
- control-flow structures
Invoking Methods

• Already saw that:

```java
System.out.println("in A");
MyClass mc = new MyClass();
mc.test();
MyClass.testStatic();
this.test();
test();
```
`instanceof`:

- instruction of the language
- Returns a boolean
- true if mc has same type or a subtype of MyClass

```
mc instanceof MyClass
```
Casts

- Used to assign an instance of a super class to a variable of a subclass
- May fail at runtime!

```java
String S = (String)myObject;
```
Blocks

• As in many languages blocks can be declared in place of any instruction and define a namespace:

```
{
    ...
}
```
if

• if (test) block/instruction;

• if (test) block/instruction; else block/instruction;

if (ms instanceof String)
    if (!ms instanceof String))
        System.out.println("MyClass not String");
    else
        System.out.println("not MyClass");
• The expression needs to return a primitive type value or one of its reference form
• break is needed

```
switch(expression){
    case constant: {}
    ...
    default: {} 
}
```
Switch Example

```java
switch(a){
    case 2: {
        System.out.println("two");
        break;
    }
    case 1: {
        System.out.println("one");
        break;
    }
    case 0: {
        System.out.println("zero");
        break;
    }
    default: {
        System.out.println(a);
    }
}
```
while

- while the test is true, it executes the next block.

```
while (test){
  ...
}
```
do...while

- executes the block once and then executes it while the test is true

```java
do{
  ...
}
while (test)
```
for

executes the *init* instruction, if *test* is true executes the block until it becomes false, at the end of each block execution executes the loop instruction

```java
for (init; test; loopInst){
    ...
}
```
for example

```java
int a[] = new int[10];
for (int i = 0; i < 10; i++) {
    System.out.println(a[i] + "\n");
}
```
Enhanced for

- if the expression is Iterable (arrays are...)

```java
for(Type varName: expression){
    ...
}
```
Enhanced for example

```java
int a[] = new int[10];
for (int v : a) {
    System.out.println(v + "\n");
}
```
break/continue

- break breaks the innermost block
- continue goes back to its start
- with labels: act on the labeled structure (continue works only on loops labels)
breaks/continue example

test: while(true){
    test2:{
        a++;
        System.out.println(a+"\n");
        if (a>10)
            break test2;
        else
            continue test;
    }
    break test;
}
Generic classes

- Declaration
- Use
- Annotations
Generic classes

Declaration

• add generic types between angle brackets:

```java
public class MyStack<E>{
    E item;
}
```
Generic Classes Use

```java
MyStack<Integer> ms = new MyStack<Integer>();
```
Generic Classes
Annotations

```java
public int count(MyClass<?> mc){...}
public int count(MyClass<? extends String> mc){...}
public int count(MyClass<? extends String & Countable> mc){...}
```
Why not subtype generics?

```java
MyStack<Integer> myStack=new MyStack<Integer>();

public static void add(Stack<Object> st, Object o){
  st.add(o);
}
```
Example: MySmallStack

```java
public void add(E value){
    Element<E> e;
    e = new Element<E>(value);
    e.next = this.first;
    this.first = e;
}

...  
MySmallStack<Integer> myStack =
    new MySmallStack<Integer>();
myStack.add(1);
```
Nested Classes

- Classes that are declared in a file, along another one
- If outside a class: static
- If inside, it depends... they can be inner classes
Inner Classes

• A particular type of nested classes
• Anonymous
• Method-scope
• Class scope
Anonymous Inner Classes

• Declared at instantiation

```java
public void wave()
    Object o = new Object()
        public Object clone()
            return this;
    }
};
...
Method Inner classes

- declared in a method
- visible only in the method

```java
// method declaration
public void wave(){
    class Hello2 extends HelloWorld{
    
    }
    ...
}
```
Class Inner Classes

- Declared in the class
- Visibility decides on access control outside of the class

```java
public class HelloWorld extends Object{
    class Hello2 extends HelloWorld{
    }
    // method declaration
    public void wave(){
    ...}
}
...
HelloWorld.Hello2 h=new HelloWorld.Hello2();
```
Some interesting classes

- Object
- String
- System
- ArrayList
Object

- See Object APIs
String

- See String APIs
System

• See System APIs
ArrayList

• See ArrayList APIs
Throwable

- The Throwable interface is meant to represent computational events that can interrupt the current computation.
- Computation can occur after the event is handled.
Exception

- Exceptions represent events that are meant to be treated.
- Whenever a method may trigger an exception, it is required that it declares so (modulo conformance). Except for RuntimeExceptions.
Runtime Exceptions

AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, EnumConstantNotFoundException, EventException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IncompleteAnnotationException, IndexOutOfBoundsException, JMRuntimeException, LSException, MalformedParameterizedTypeException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, RejectedExecutionException, SecurityException, SystemException, TypeNotPresentException, UndeclaredThrowableException, UnmodifiableSetException,

UnsupportedOperationException ...
Error

- Meant to represent an unrecoverable error
- Can be recovered still...

- Example: AnnotationFormatError, AssertionError, AWTError, CoderMalfunctionError, FactoryConfigurationException, LinkageError, ThreadDeath, TransformerFactoryConfigurationException, VirtualMachineError
Throw, throws

- it is possible to throw an exception manually by using:
  \[ \text{throw an\_exception;} \]

- methods that may fail due to an exception (non-runtime) have to indicate it:

  \[ \text{public void m() throws MyException{...}} \]